

# **THE CIP NETWORKS LIBRARY**

## **Volume 1**

### **Common Industrial Protocol (CIP<sup>TM</sup>)**

---

Edition 3.3

November, 2007

ODVA & ControlNet International Ltd.

The CIP Networks Library  
Volume 1: Common Industrial Protocol (CIP™)

Publication Number: PUB00001

Copyright © 2001 through 2007 Open DeviceNet Vendor Association, Inc. (ODVA). All rights reserved. For permissions to reproduce excerpts of this material, with appropriate attribution to the author(s), please contact ODVA at:

Open DeviceNet Vendor Association, Inc.  
4220 Varsity Drive, Suite A, Ann Arbor, MI 48108-5006 USA  
TEL                1-734-975-8840  
FAX                1-734-922-0027  
EMAIL              odva@odva.org  
WEB                www.odva.org

The right to make, use or sell product or system implementations described herein is granted only under separate license pursuant to a Terms of Usage Agreement or other agreement. Terms of Usage Agreements for individual CIP Networks are available, at standard charges, over the Internet at the following web sites:

**[www.odva.org](http://www.odva.org)** - Terms of Usage Agreements for CompoNet, DeviceNet, EtherNet/IP and CompoNet along with general information on CIP Networks and the association of ODVA

**[www.controlnet.org](http://www.controlnet.org)** - Terms of Usage Agreement for ControlNet along with general information on ControlNet and ControlNet International.

#### Warranty Disclaimer Statement

Because CIP Networks may be applied in many diverse situations and in conjunction with products and systems from multiple vendors, the user and those responsible for specifying CIP Networks must determine for themselves its suitability for the intended use. The Specifications are provided to you on an AS IS basis without warranty. **NO WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING WITHOUT LIMITATION ANY WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE ARE BEING PROVIDED BY THE PUBLISHER, ODVA AND/OR CONTROLNET INTERNATIONAL.** In no event shall the Publisher, ODVA and/or ControlNet International, their officers, directors, members, agents, licensors or affiliates be liable to you or any customer for lost profits, development expenses or any other direct, indirect incidental, special or consequential damages.

ControlNet and ControlNet CONFORMANCE TESTED are trademarks of ControlNet International, Ltd.

CIP, DeviceNet, DeviceNet CONFORMANCE TESTED, DeviceNet Safety, DeviceNet Safety CONFORMANCE TESTED, CompoNet and CompoNet CONFORMANCE TESTED, EtherNet/IP CONFORMANCE TESTED, EtherNet/IP Safety CONFORMANCE TESTED, and CIP Safety are trademarks of Open DeviceNet Vendor Association, Inc.

EtherNet/IP is a trademark of ControlNet International under license by Open DeviceNet Vendor Association, Inc.

All other trademarks referenced herein are property of their respective owners.

# The CIP Networks Library: Volume 1

## Common Industrial Protocol (CIP<sup>TM</sup>)

### Table of Contents

<b>Revisions</b>	- Summary of Changes in this Edition
<b>Preface</b>	- Organization of CIP Networks Specifications - The Specification Enhancement Process
<b>Chapter 1</b>	- Introduction to the Common Industrial Protocol
<b>Chapter 2</b>	- Messaging Protocol
<b>Chapter 3</b>	- Communications Objects
<b>Chapter 4</b>	- CIP Object Model
<b>Chapter 5</b>	- Object Library
<b>Chapter 6</b>	- Device Profiles
<b>Chapter 7</b>	- Electronic Data Sheets
<b>Chapter 8</b>	- Physical Layer
<b>Chapter 9</b>	- Indicators and Middle Layers
<b>Chapter 10</b>	- Bridging and Routing
<b>Appendix A</b>	- Explicit Messaging Services
<b>Appendix B</b>	- Status Codes
<b>Appendix C</b>	- Data Management
<b>Appendix D</b>	- Engineering Units

This page is intentionally left blank

## Revisions

The CIP Networks Library Volume 1: The Common Industrial Protocol Edition 3.3 contains the following changes from Edition 3.2. Please see the change bars on the pages noted here for specific modifications. Note: Some of the pages within the ranges noted may not contain any changes.

Chapter	Pages	Reason for Change
<b>Add new extended status code and definition of scheduled network</b>		
1-7.47	1-19	<ul style="list-style-type: none"> <li>• Add new definition to table</li> </ul>
3-5.6.1	3-80	<ul style="list-style-type: none"> <li>• Add new extended error code to Table 3-5.29</li> </ul>
<b>Connection Path Enhancement</b>		
3-5.5.1.10	3-63	<ul style="list-style-type: none"> <li>• Modify three cells in Table 3-5.13</li> </ul>
C-1.5	C-14	<ul style="list-style-type: none"> <li>• Add new hierarchy example</li> </ul>
<b>AssemN Optional Size Field</b>		
7-3.6.7.2	7-49	<ul style="list-style-type: none"> <li>• Change Size from Required to Optional in Table 7-3.24</li> </ul>
7-3.6.7.2.3	7-50	<ul style="list-style-type: none"> <li>• Add new paragraphs to describe when the size field is not present</li> </ul>
<b>Allow 32-bit Logical Instances and Connection Points</b>		
C-1.4.2	C-8	<ul style="list-style-type: none"> <li>• Modify 32-bit logical address to indicate when allowed for use</li> </ul>
5-5	5-43	<ul style="list-style-type: none"> <li>• Change Assembly Instance ID ranges in Table 5-5.2</li> </ul>
5-42	5-351	<ul style="list-style-type: none"> <li>• Change reserved instance ID range in Table 5-42.1</li> </ul>
<b>Assembly Object Member Service Support</b>		
5-5.3	5-45	<ul style="list-style-type: none"> <li>• Update Description of Service field for Get_Member &amp; Set_Member services and make Set_Member service optional at the instance level.</li> </ul>
<b>Drop ControlNet from the ControlNet Programmable Controller Profile</b>		
6-7	6-21	<ul style="list-style-type: none"> <li>• Remove ControlNet from the name in Table 6-7.1</li> </ul>
6-33	6-180	<ul style="list-style-type: none"> <li>• Remove ControlNet from the name of the profile</li> <li>• Remove ControlNet and remove reference to scheduled originator from first paragraph</li> </ul>
6-33.1	6-180	<ul style="list-style-type: none"> <li>• Modify caption of Table 6-33.1, remove Scheduling Object and remove ControlNet from last paragraph</li> </ul>
6-33.2	6-18	<ul style="list-style-type: none"> <li>• Remove ControlNet from first paragraph and remove Scheduling Object from Table 6-33.2</li> </ul>
<b>Add Enhanced Mass Flow Controller Device Profile</b>		
6-7	6-21	<ul style="list-style-type: none"> <li>• Add new device to Device Type list</li> </ul>
6-42	6-292	<ul style="list-style-type: none"> <li>• Add new section.</li> </ul>
<b>Add UCD-4 Unicode Encoding Support to STRINGI</b>		
C-2.1.2	C-18	<ul style="list-style-type: none"> <li>• Add last line to Table C-2.3</li> </ul>
<b>Connection Configuration Object Get_Attributes_All class level response data</b>		
5-48.4	5-567	<ul style="list-style-type: none"> <li>• Change Set_Attributes_All to N/A for class level</li> </ul>
5-48.4.1	5-567	<ul style="list-style-type: none"> <li>• Add new text and table for Get All response – Class level</li> </ul>
5-48.4.2	5-569	<ul style="list-style-type: none"> <li>• Add new text to first paragraph and changes to Table 5-48.12</li> </ul>
5-48.4.3	5-571, 572	<ul style="list-style-type: none"> <li>• Add new text to first paragraph and changes to Table 5-48.15</li> </ul>
5-48.4.4	5-572	<ul style="list-style-type: none"> <li>• Add new text to first paragraph and changes to Table 5-48.16</li> </ul>
5-48.4.5	5-572	<ul style="list-style-type: none"> <li>• Add new text to first paragraph and changes to Table 5-48.17</li> </ul>
5-48.4.7	5-573	<ul style="list-style-type: none"> <li>• Remove Error Table</li> </ul>
5-48.4.8	5-573	<ul style="list-style-type: none"> <li>• Remove Error Table</li> </ul>
5-48.4.9	5-573	<ul style="list-style-type: none"> <li>• Remove Error Table</li> </ul>
5-48.4.10	5-574	<ul style="list-style-type: none"> <li>• Remove Error Table</li> </ul>
5-48.4.11	5-574	<ul style="list-style-type: none"> <li>• Add new text to first paragraph and changes to Table 5-48.23</li> </ul>
5-48.4.12	5-574	<ul style="list-style-type: none"> <li>• Add new text to first paragraph and changes to Table 5-48.26</li> </ul>
5-48.4.13	5-575, 576	<ul style="list-style-type: none"> <li>• Add new text to first paragraph and changes to Table 5-48.28</li> </ul>
5-48.4.14	5-576	<ul style="list-style-type: none"> <li>• Add new text to first paragraph and changes to Table 5-48.30</li> </ul>

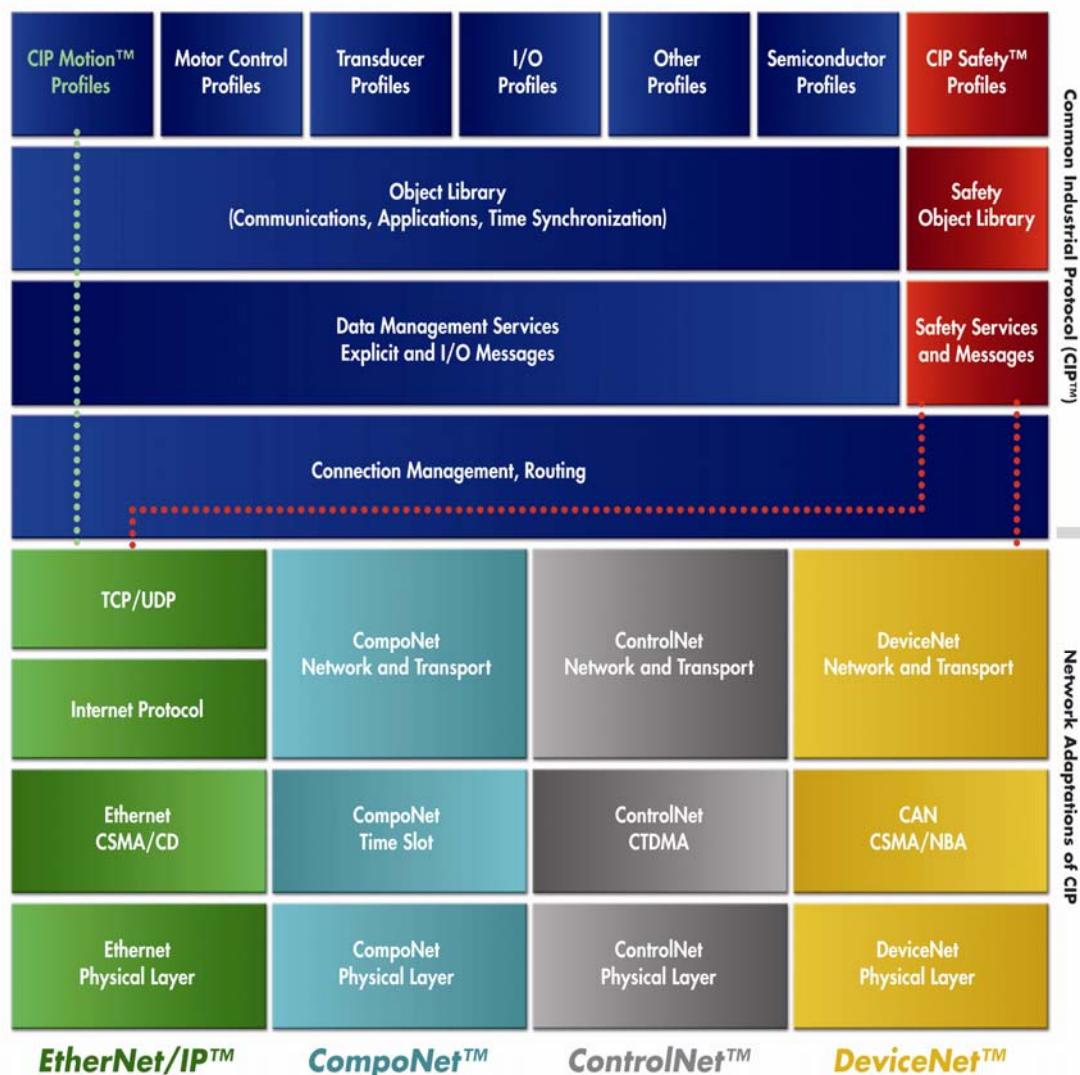
<b>CompoNet Hooks</b>		
3-7.3	3-89	<ul style="list-style-type: none"> <li>• Change Port Type attribute access from Set to Get</li> <li>• Add value 200 = CompoNet</li> </ul>
5-1	5-22	<ul style="list-style-type: none"> <li>• Add 2 CompoNet objects to Table 5-5.1</li> </ul>
5-46.1	5-403	<ul style="list-style-type: none"> <li>• Revise wording to allow use with any CIP network</li> </ul>
5-47.9	5-547	<ul style="list-style-type: none"> <li>• Add CompoNet to Table 5-47.1</li> </ul>
5-47.13.1.10.1	5-552	<ul style="list-style-type: none"> <li>• Add CompoNet to Table 5-47.5</li> </ul>
5-47.13.1.10.2	5-552	<ul style="list-style-type: none"> <li>• Add CompoNet to Table 5-47.6</li> </ul>
6-2.1	6-13	<ul style="list-style-type: none"> <li>• Numerous changes to Table 6-2.2</li> </ul>
6-7	6-20	<ul style="list-style-type: none"> <li>• Add CompoNet Repeater to the Device Profile list in Table 6-7.1</li> </ul>
7-3.6.4	7-44	<ul style="list-style-type: none"> <li>• Add CompoNet to the classification list</li> </ul>
7-3.6.10	7-70	<ul style="list-style-type: none"> <li>• Add CompoNet to the Port Type list</li> </ul>
8-1	8-3	<ul style="list-style-type: none"> <li>• Remove examples of CIP networks (last sentence)</li> </ul>
9-1	9-3	<ul style="list-style-type: none"> <li>• Remove examples of CIP networks (last sentence)</li> </ul>
B-1	B-4	<ul style="list-style-type: none"> <li>• Define status codes 23 and 24 (previously reserved)</li> </ul>
<b>Get_All/Set_All Service Data Size Rules and Related Issues</b>		
4-4	4-4	<ul style="list-style-type: none"> <li>• Move definition of the term “default” from this section to 4-4.1</li> </ul>
4-4.1	4-5	<ul style="list-style-type: none"> <li>• Add clarification for default value</li> </ul>
4-5.1	4-9	<ul style="list-style-type: none"> <li>• Numerous changes</li> </ul>
4-5.2	4-9, 10	<ul style="list-style-type: none"> <li>• Remove words in first paragraph and replace last paragraph</li> </ul>
4-9.2	4-19	<ul style="list-style-type: none"> <li>• Numerous changes</li> </ul>
4-9.2.1.1	4-21	<ul style="list-style-type: none"> <li>• Numerous changes to items 1 and 2</li> </ul>
4-9.2.1.2	4-22	<ul style="list-style-type: none"> <li>• Removed subsections 4-9.2.1.2.2 (entirely) and 4-9.2.1.2.3 (partially)</li> </ul>
A-4.1	A-7	<ul style="list-style-type: none"> <li>• Clarified wording of what is returned in the service reply</li> </ul>
A-4.1.1	A-7, 8	<ul style="list-style-type: none"> <li>• Numerous changes</li> </ul>
A-4.2.1	A-9	<ul style="list-style-type: none"> <li>• Numerous changes</li> </ul>
<b>EF Protocol Safety Hooks</b>		
5-48.2	5-562	<ul style="list-style-type: none"> <li>• Add three new instance attributes (20-22) to Table 5-48.3</li> </ul>
5-48.4.1	5-569	<ul style="list-style-type: none"> <li>• Removed names of Safety Parameters, corrected capitalization and added additional safety parameters to end of Get_Attributes_All Response Data</li> </ul>
5-48.4.2	5-571	<ul style="list-style-type: none"> <li>• Removed names of Safety Parameters, corrected capitalization and added additional safety parameters to end of Set_Attributes_All Response Data</li> </ul>
<b>Modbus Integration Hooks</b>		
3-7.3	3-89, 90	<ul style="list-style-type: none"> <li>• Add new protocols to Attribute 1 and change TCP/IP to EtherNet/IP, change Description &amp; Semantics of attribute 4 in Table 3-7.3</li> </ul>
5-1	5-22	<ul style="list-style-type: none"> <li>• Add Modbus Object to list of defined objects</li> </ul>
5-2.2	5-24	<ul style="list-style-type: none"> <li>• Add attribute 18 to Identity Object instance attribute list</li> </ul>
5-14.2.1.1	5-99	<ul style="list-style-type: none"> <li>• Modify Bits 4 &amp; 5 description, add bit 14 definition in Table 5-14.9</li> </ul>
6-7	6-21	<ul style="list-style-type: none"> <li>• Add CIP Modbus device type to list of defined devices</li> </ul>
7-3.6.4	7-44	<ul style="list-style-type: none"> <li>• Add ModbusSL &amp; ModbusTCP to Device Classifications list</li> </ul>
7-3.6.5	7-45	<ul style="list-style-type: none"> <li>• Changed Data Type of Scaling Offset and added footnote, both in Table 7-3.22</li> </ul>
7-3.6.10	7-69	<ul style="list-style-type: none"> <li>• Changed Port name to Conditional and added footnote, both in Table 7-3.38</li> <li>• Added ModbusSL &amp; ModbusTCP to port types list</li> <li>• Modified description of Port Name field</li> </ul>
B-1	B-4	<ul style="list-style-type: none"> <li>• Define General Status Code 2B</li> </ul>
<b>Typographical errors/corrections</b>		
3-5.5.1.10.1	3-65	<ul style="list-style-type: none"> <li>• Remove extra “the” from the first sentence</li> </ul>
3-6.1.4	3-83	<ul style="list-style-type: none"> <li>• Corrected caption of Table 3-6.1 to say “32-bit...”</li> </ul>
5-12.5.1.2	5-88	<ul style="list-style-type: none"> <li>• Changed “Discrete” to “Analog”</li> </ul>
5-48.2.2.1	5-563	<ul style="list-style-type: none"> <li>• Corrected reference to a table in Chapter 3. Table was no longer at that reference due to revisions to the chapter. Changed reference to something more distinct and less likely to change over time.</li> </ul>
5-48.2.2.4	5-564	<ul style="list-style-type: none"> <li>• Removed extra word (“a”) from the second sentence of the second paragraph</li> </ul>
C-all	-	<ul style="list-style-type: none"> <li>• Replaced all figure/table captions and references</li> </ul>

## Preface

### Organization of the CIP Networks Specifications

Today, four networks - DeviceNet™, ControlNet™, EtherNet/IP™ and CompoNet™ - use the Common Industrial Protocol (CIP) for the upper layers of their network protocol. For this reason, the associations that manage these networks - ODVA and ControlNet International - have mutually agreed to manage and distribute the specifications for CIP Networks in a common structure to help ensure consistency and accuracy in the management of these specifications.

The following diagram illustrates the organization of the library of CIP Network specifications. In addition to CIP Networks, CIP Safety™ consists of the extensions to CIP for functional safety.



This common structure presents CIP in one volume with a separate volume for each network adaptation of CIP. The specifications for the CIP Networks are two-volume sets, paired as shown below.

The EtherNet/IP specification consists of:

Volume 1: Common Industrial Protocol (CIP™)

Volume 2: EtherNet/IP Adaptation of CIP

The DeviceNet specification consists of:

Volume 1: Common Industrial Protocol (CIP™)

Volume 3: DeviceNet Adaptation of CIP

The ControlNet specification consists of:

Volume 1: Common Industrial Protocol (CIP™)

Volume 4: ControlNet Adaptation of CIP

The CompoNet specification consists of:

Volume 1: Common Industrial Protocol (CIP™)

Volume 6: CompoNet Adaptation of CIP

The specification for CIP Safety™ is distributed in a single volume:

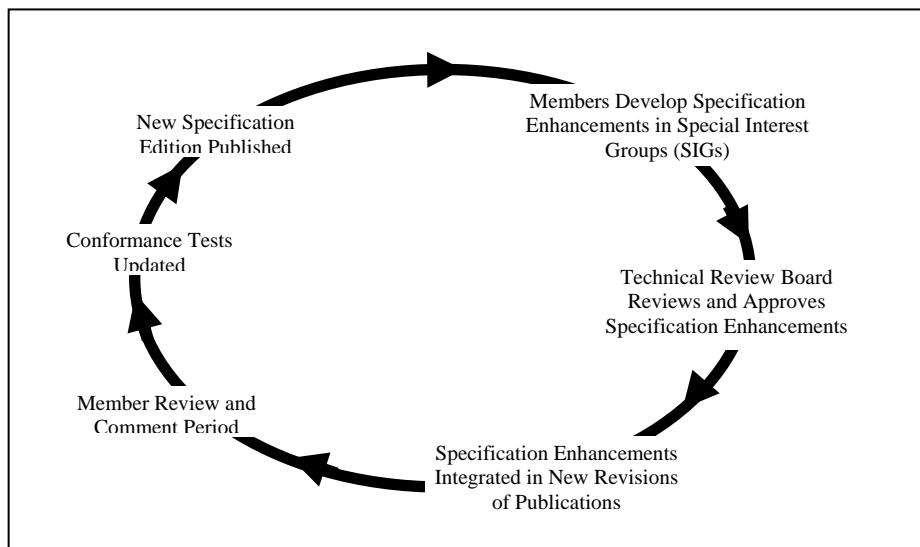
Volume 5: CIP Safety

The specification for integrating Modbus Devices is distributed in a single volume:

Volume 7: Integration of Modbus Devices into the CIP Architecture

#### Specification Enhancement Process

The specifications for CIP Networks are continually being enhanced to meet the increasing needs of users for features and functionality. ODVA and ControlNet International have also agreed to operate using a common Specification Enhancement Process in order to ensure open and stable specifications for all CIP Networks. This process is on going throughout the year for each CIP Network Specification as shown in the figure below. New editions of each CIP Network specification are published on a periodic basis.



# **Volume 1: Common Industrial Protocol (CIP<sup>TM</sup>) Specification**

## **Chapter 1: Introduction to CIP**

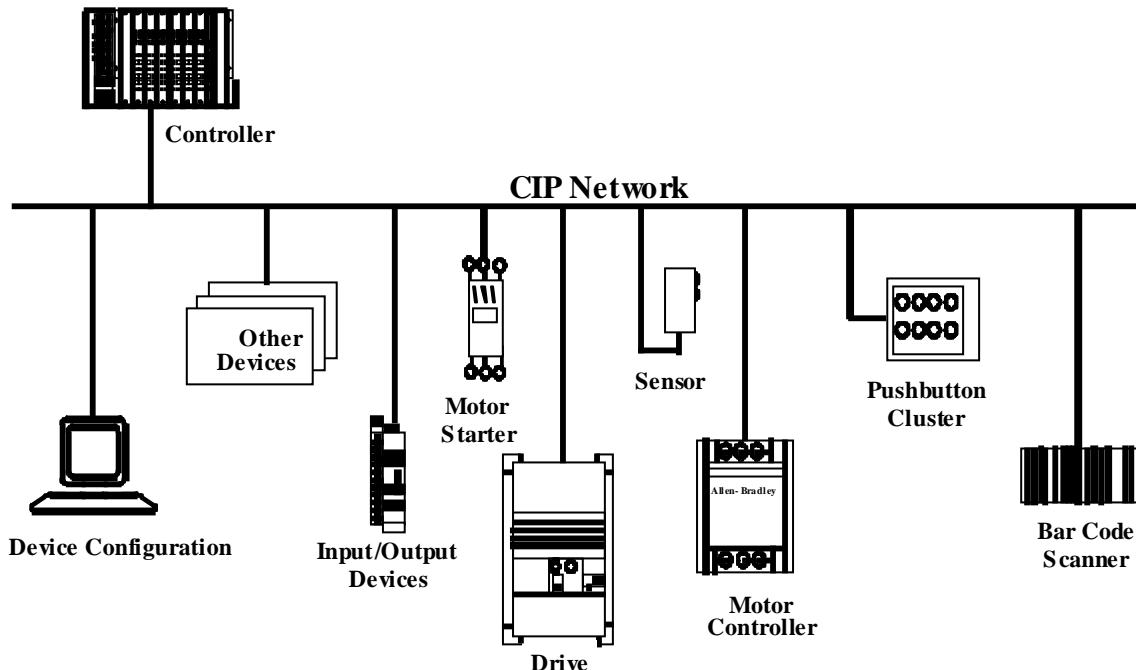
## Contents

1-1	Introduction.....	3
1-2	Object Modeling .....	4
1-2.1	Object Addressing.....	6
1-2.2	Address Ranges.....	9
1-3	Network Overview.....	10
1-3.1	I/O Connections .....	11
1-3.2	Explicit Messaging Connections.....	12
1-4	CIP Object Model .....	13
1-5	System Structure .....	14
1-5.1	Topology.....	14
1-5.2	Logical Structure.....	15
1-6	CIP Specification Structure.....	16
1-7	Definitions .....	17
1-8	Abbreviations.....	20

## 1-1 Introduction

The Common Industrial Protocol (CIP) is a peer to peer object oriented protocol that provides connections between industrial devices (sensors, actuators) and higher-level devices (controllers). CIP is physical media and data link layer independent. See Figure 1-1.1.

**Figure 1-1.1 Example CIP Communication Link**



CIP has two primary purposes:

- Transport of control-oriented data associated with I/O devices
- Transport of other information that is related to the system being controlled, such as configuration parameters and diagnostics.

## 1-2 Object Modeling

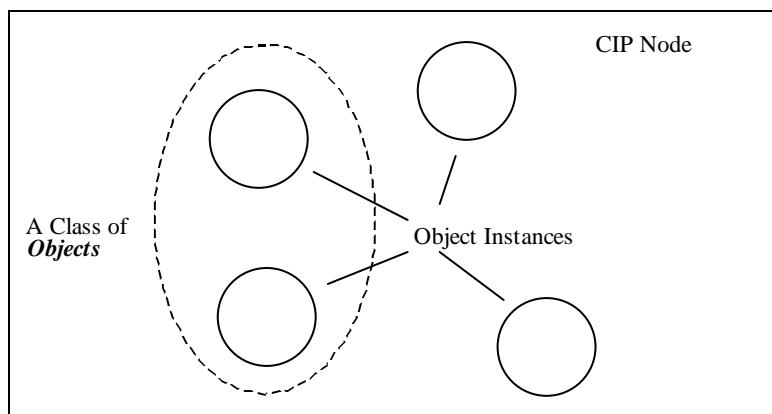
CIP makes use of abstract *object modeling* to describe:

- The suite of communication services available
- The externally visible behavior of a CIP node
- A common means by which information within CIP products is accessed and exchanged

A CIP node is modeled as a collection of *Objects*. An **Object** provides an abstract representation of a particular component within a product. The realization of this abstract object model within a product is implementation dependent. In other words, a product internally maps this object model in a fashion specific to its implementation.

A **Class** is a set of Objects that all represent the same kind of system component. An **Object Instance** is the actual representation of a particular Object within a Class. Each *Instance* of a Class has the same set of attributes, but has its own particular set of attribute values. As Figure 1-2.1 illustrates, multiple *Object Instances* within a particular Class can reside in a CIP node.

**Figure 1-2.1 A Class of Objects**



An Object Instance and/or an Object Class has *Attributes*, provides *Services*, and implements a *Behavior*.

**Attributes** are characteristics of an Object and/or an Object Class. Typically, Attributes provide status information or govern the operation of an Object. **Services** are invoked to trigger the Object/Class to perform a task. The **Behavior** of an Object indicates how it responds to particular events. For example, a person can be abstractly viewed as an Instance within the Class *Human*. Generally speaking, all humans have the same set of attributes: age, gender, etc., yet, because the values of each attribute vary, each of us looks/behave in a distinct fashion.

**Table 1-2.1 Object Model Terminology Examples**

Class	Instances	Attributes	Attribute Values
Human	Mary	Gender	Female
		Age	31
	Jerry	Gender	Male
		Age	50

The following Object Modeling related terms are used when describing CIP services and protocol.

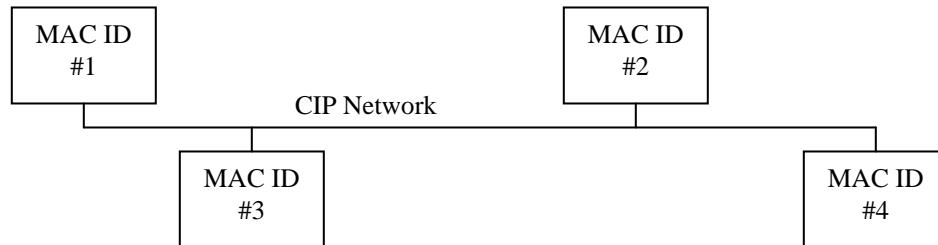
- **Object** – An abstract representation of a particular component within a product.
- **Class** – A set of objects that all represent the same kind of system component. A class is a generalization of an object. All objects in a class are identical in form and behavior, but may contain different attribute values.
- **Instance** – A specific and real (physical) occurrence of an object. For example: New Zealand is an instance of the object class Country. The terms Object, Instance, and Object Instance all refer to a specific Instance.
- **Attribute** – A description of an externally visible characteristic or feature of an object. Typically, attributes provide status information or govern the operation of an Object. For example: the ASCII name of an object; and the repetition rate of a cyclic object.
- **Instantiate** - To create an instance of an object with all instance attributes initialized to zero unless default values are specified in the object definition.
- **Behavior** – A specification of how an object acts. Actions result from different events the object detects, such as receiving service requests, detecting internal faults or elapsing timers.
- **Service** – A function supported by an object and/or object class. CIP defines a set of common services and provides for the definition of Object Class and/or Vendor Specific services. CIP common services are those whose parameters and required behaviors are defined in Appendix A.
- **Communication Objects** - A reference to the Object Classes that manage and provide the run-time exchange of implicit (I/O) and explicit messages.
- **Application Objects** - A reference to multiple Object Classes that implement product-specific features.

## 1-2.1 Object Addressing

The information in this section provides a common basis for logically addressing separate physical components across CIP. The following list describes the information that is used to address an Object from a CIP network:

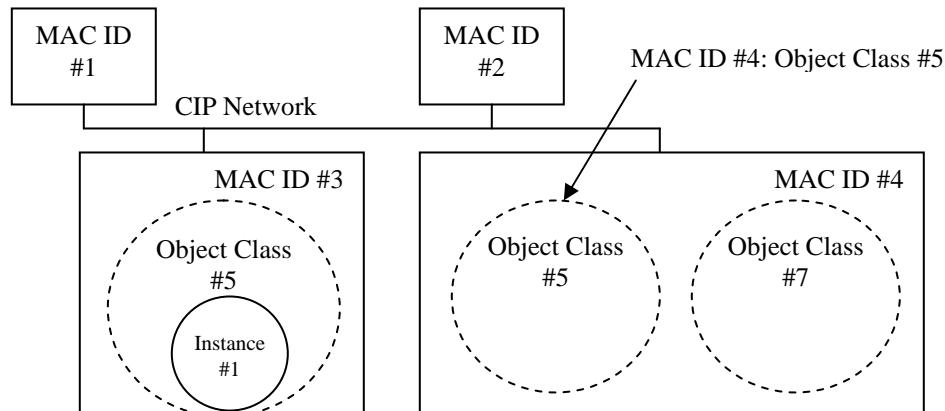
- **Media Access Control Identifier (MAC ID)** - An identification value assigned to each node on the CIP network. This value distinguishes a node among all other nodes on the same link. The MAC ID format is network specific.

**Figure 1-2.2 MAC ID Example**



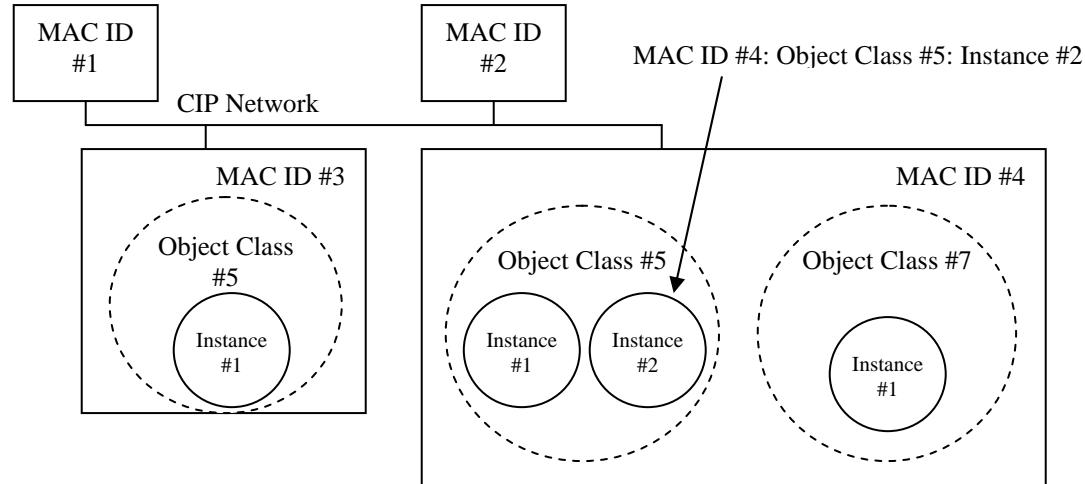
- **Class Identifier (Class ID)** - An integer identification value assigned to each Object Class accessible from the network.

**Figure 1-2.3 Class ID Example**



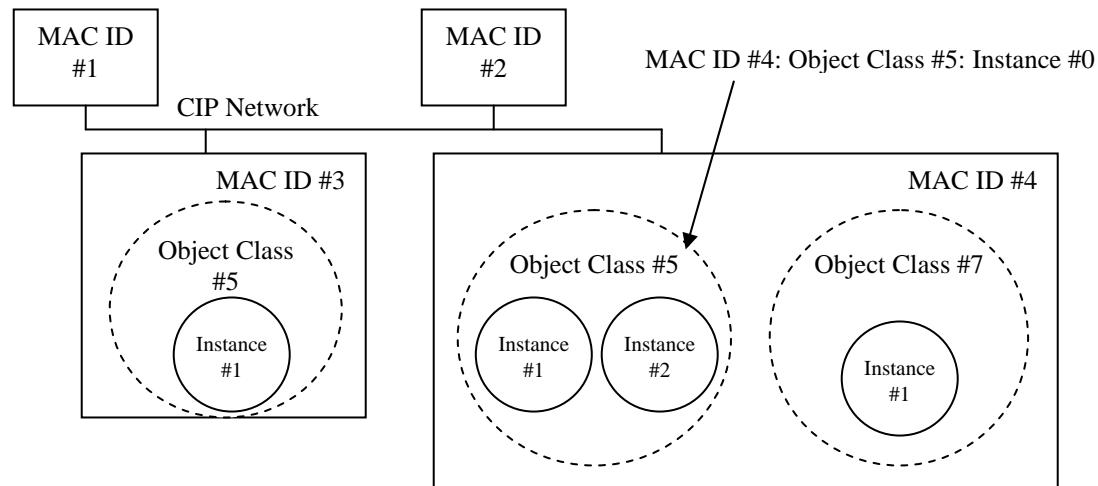
- **Instance Identifier (Instance ID)** - An integer identification value assigned to an Object Instance that identifies it among all Instances of the same Class. This integer is unique within the MAC ID:Class in which it resides.

**Figure 1-2.4 Instance Identifier Example**



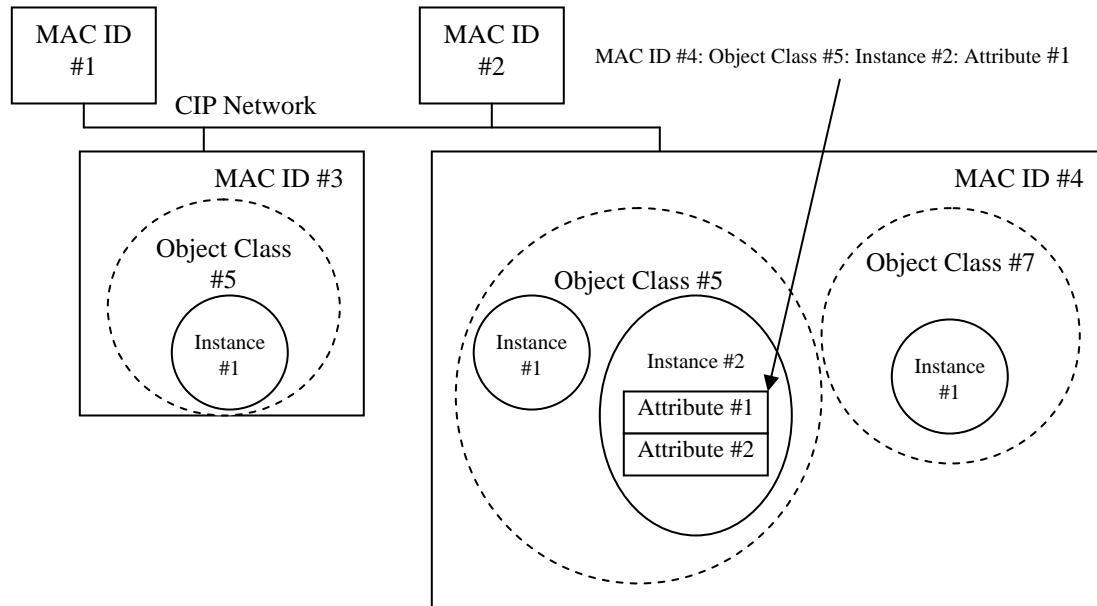
It is also possible to address the Class itself versus a specific Object Instance within the Class. This is accomplished by utilizing the Instance ID value zero (0). **CIP reserves the Instance ID value zero (0) to indicate a reference to the Class versus a specific Instance within the Class.**

**Figure 1-2.5 Instance #0 Example**



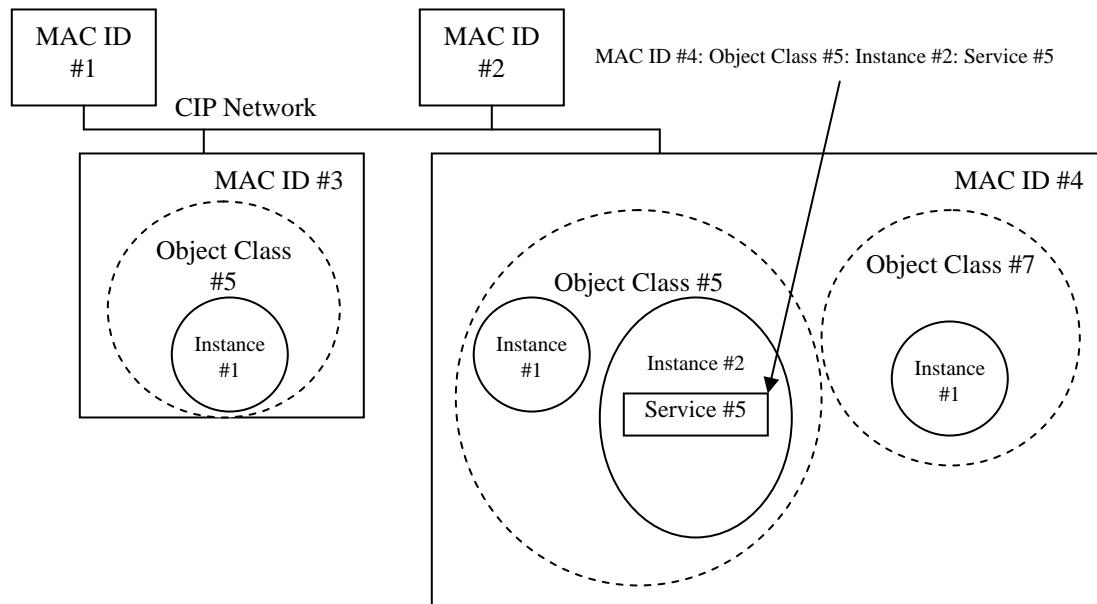
- **Attribute Identifier (Attribute ID)** - An integer identification value assigned to a Class and/or Instance Attribute.

**Figure 1-2.6 Attribute ID Example**



- **Service Code** - An integer identification value which denotes a particular Object Instance and/or Object Class function.

**Figure 1-2.7 Service Code Example**



## 1-2.2 Address Ranges

This section presents CIP defined ranges for the Object Addressing information presented in the previous section. The following terms are used when defining the ranges:

- **Open** - A range of values whose meaning is defined by ODVA/CI and are common to all CIP participants.
- **Vendor Specific** - A range of values specific to the vendor of a device. These are used by vendors to extend their devices beyond the available Open options. A vendor internally manages the use of values within this range.
- **Object Class Specific** - A range of values whose meaning is defined by an Object Class. This range applies to Service Code definitions.

Table 1-2.2 defines the ranges applicable to Class ID values.

**Table 1-2.2 Class ID Ranges**

Range	Meaning
00 - 63 <sub>hex</sub>	CIP Common
64 <sub>hex</sub> - C7 <sub>hex</sub>	Vendor Specific
C8 <sub>hex</sub> - EF <sub>hex</sub>	Reserved by ODVA/CI for future use
F0 <sub>hex</sub> - 2FF <sub>hex</sub>	CIP Common
300 <sub>hex</sub> - 4FF <sub>hex</sub>	Vendor Specific
500 <sub>hex</sub> - FFFF <sub>hex</sub>	Reserved by ODVA/CI for future use

Table 1-2.3 defines the ranges applicable to Service Code values.

**Table 1-2.3 Service Code Ranges**

Range	Meaning
00 - 31 <sub>hex</sub>	CIP Common. These are referred to as <i>CIP Common Services</i> . These are defined in Appendix A, Explicit Messaging Services.
32 <sub>hex</sub> - 4A <sub>hex</sub>	Vendor Specific
4B <sub>hex</sub> - 63 <sub>hex</sub>	Object Class Specific
64 <sub>hex</sub> - 7F <sub>hex</sub>	Reserved by ODVA/CI for future use
80 <sub>hex</sub> - FF <sub>hex</sub>	Invalid/Not used

Table 1-2.4 defines the ranges applicable to Attribute ID values.

**Table 1-2.4 Attribute ID Ranges**

Range	Meaning
00 - 63 <sub>hex</sub>	CIP Common
64 <sub>hex</sub> - C7 <sub>hex</sub>	Vendor Specific
C8 <sub>hex</sub> - FF <sub>hex</sub>	Reserved by ODVA/CI for future use
100 <sub>hex</sub> - 2FF <sub>hex</sub>	CIP Common
300 <sub>hex</sub> - 4FF <sub>hex</sub>	Vendor Specific
500 <sub>hex</sub> - 8FF <sub>hex</sub>	CIP Common
900 <sub>hex</sub> - CFF <sub>hex</sub>	Vendor Specific
D00 <sub>hex</sub> - FFFF <sub>hex</sub>	Reserved by ODVA/CI for future use

## 1-3 Network Overview

CIP defines a connection-based scheme to facilitate all application communications. A CIP **connection** provides a communication path between multiple end-points. The end-points of a connection are applications that need to share data. Transmissions associated with a particular connection are assigned an identification value when a connection is established. This identification value is called the **Connection ID (CID)**.

**Connection Objects** model the communication characteristics of a particular Application-to-Application(s) relationship. The term **end-point** refers to one of the communicating entities involved in a connection.

CIP's connection-based scheme defines a dynamic means by which the following two types of connections can be established:

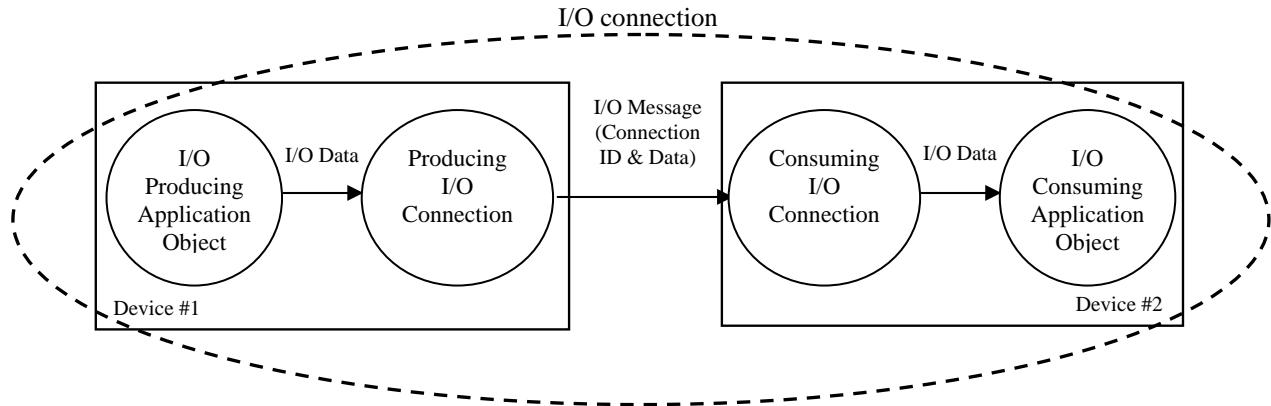
- **I/O Connections** - Provide dedicated, special-purpose communication paths between a producing application and one or more consuming applications for the purpose of moving application-specific data. This is often referred to as implicit messaging
- **Explicit Messaging Connections** - Provide generic, multi-purpose communication paths between two devices. These connections often are referred to as just Messaging Connections. Explicit Messages provide the typical request/response-oriented network communications.

### 1-3.1 I/O Connections

As previously stated, I/O *Connections* provide special-purpose communication paths between a producing application and one or more consuming applications. Application-specific I/O data moves across an I/O Connection.

**I/O Messages** are exchanged across I/O connections. An I/O Message consists of a Connection ID and optional associated I/O data. The meaning of the data within an I/O Message is *implied* by the associated Connection ID. The connection end-points are assumed to have knowledge of the intended use or meaning of the I/O Message.

**Figure 1-3.1 CIP I/O Connection**



**This document does not define any particular use for I/O Messaging.** There are a wide variety of functions that can be accomplished using I/O Messaging. Either by virtue of the particular type of product transmitting an I/O Message, or based upon configuration performed using Explicit Messaging, the meaning and/or intended use of all I/O Messages can be made known to the system. See Chapter 6 Device Profiles for I/O message formats defined for common device types.

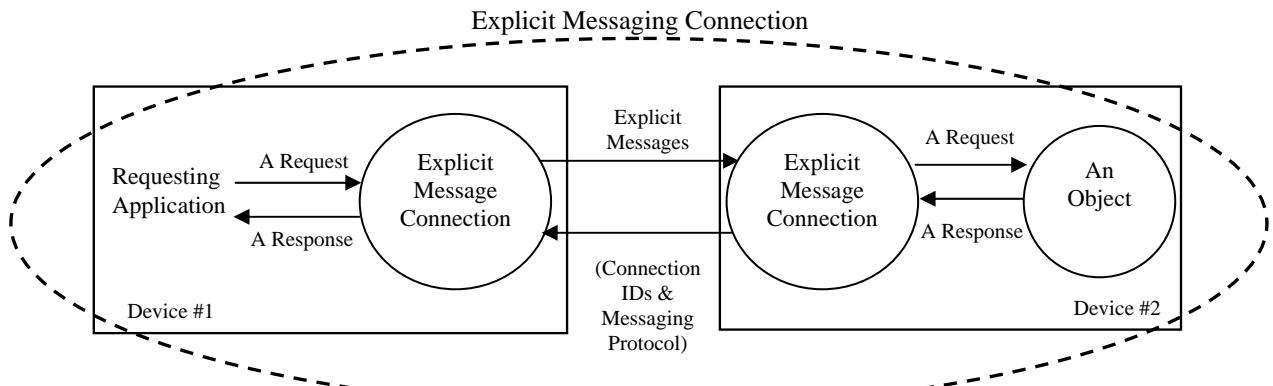
### 1-3.2 Explicit Messaging Connections

Explicit Messaging Connections provide generic, multi-purpose communication paths between two devices. Explicit Messages are exchanged across Explicit Messaging Connections.

**Explicit Messages** are used to command the performance of a particular task and to report the results of performing the task. Explicit Messaging provides the means by which typical request/response oriented functions are performed (e.g. module configuration).

CIP defines an Explicit Messaging protocol that states the meaning of the message. An Explicit Message consists of a Connection ID and associated messaging protocol information.

**Figure 1-3.2 CIP Explicit Messaging Connection**

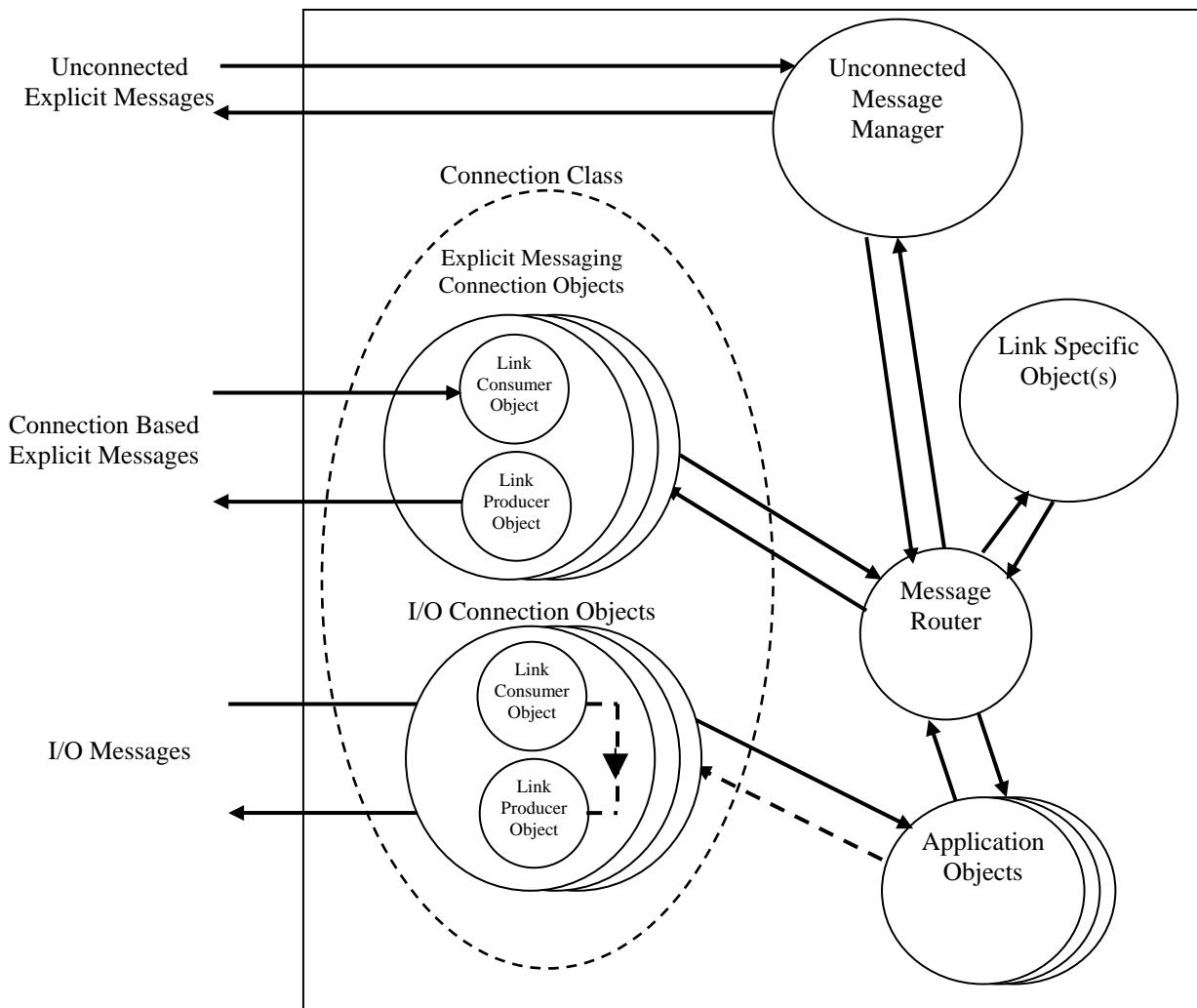


1-4 CIP Object Model

Figure 1-4.1 illustrates the abstract object model of a CIP product. Included are the following components:

- **Unconnected Message Manager (UCMM)** - Processes CIP Unconnected Explicit messages.
  - **Connection Object Class**- Allocates and manages internal resources associated with both I/O and Explicit Messaging connections.
  - **Connection Object Instance**- Manages the communication-specific aspects associated with a particular application-to-application network relationship.
  - **Connection Manager** – An object class, required in some CIP network nodes, used to control certain aspects of Connection Object instances within the node.
  - **Network-Specific Link Object** - Provides the configuration and status of a physical CIP network connection (eg. DeviceNet and ControlNet objects).
  - **Message Router** - Distributes Explicit Request Messages to the appropriate handler object.
  - **Application Objects** - Implement the intended purpose of the product.

### **Figure 1-4.1 CIP Device Object Model**



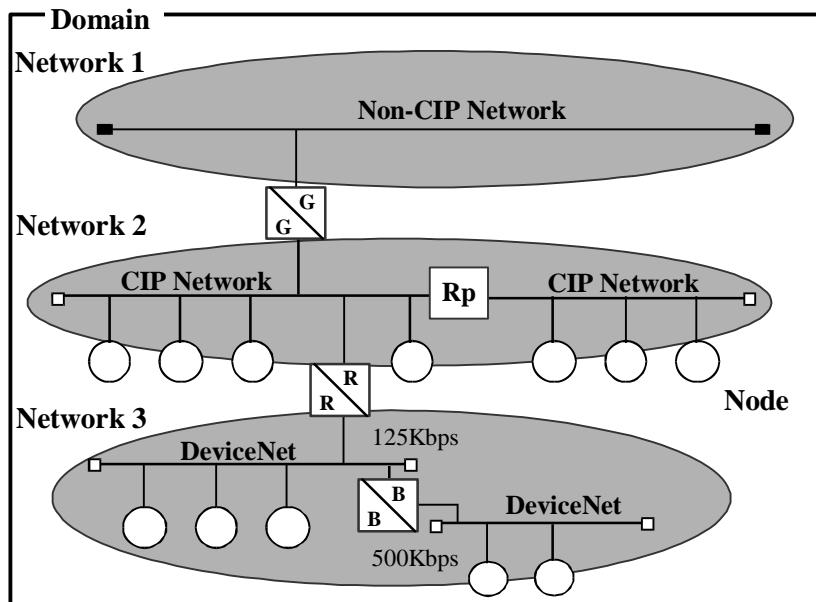
## 1-5 System Structure

### 1-5.1 Topology

The system structure uses the following physical organization.

- **System = { Domain(s) }**
  - System contains one or more domains.
- **Domain = { Network(s) }**
  - A domain is a collection of one or more networks. Networks must be unique within a domain. A domain may contain a variety of network types.
- **Network = { Subnet(s) }**
  - A network is a collection of one or more subnets, where each node's MAC ID is unique on the network.
- **Subnet = { Nodes(s) }**
  - A subnet is a collection of nodes using a common protocol and shared media access arbitration. i.e. subnet may have multiple physical segments and contain repeaters.
- **Segment = { Nodes(s) }**
  - A segment is a collection of nodes connected to a single uninterrupted section of physical media.
- **Node = { Object(s) }**
  - A node is a collection of objects that communicate over a subnet, and arbitrates using a single MAC ID. A physical device may contain one or more nodes.

**Figure 1-5.1 System Structure - Topology**



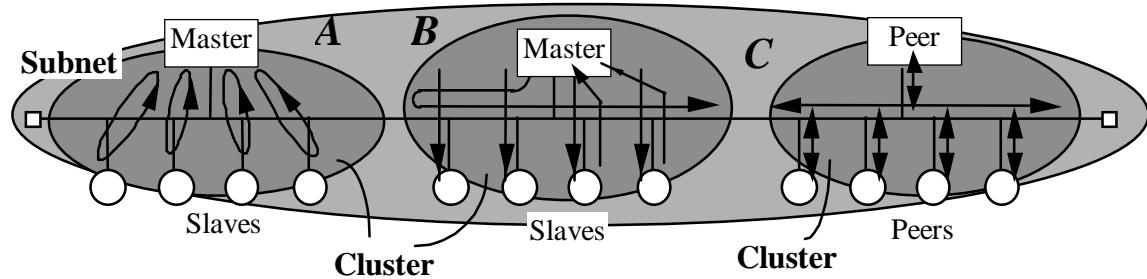
-  Segment Repeater between segments
  - Segments participate in the same media arbitration
-  Subnet Bridge between subnets
  - Duplicate MAC ID check passes through
  - MAC ID's on one subnet shall not be duplicated on the other subnet
  - Subnets may operate at different baud rates
-  Network Router between similar networks
  - Both networks are CIP-based
-  Gateway between dissimilar networks
  - One network is CIP-based, the other is not

## 1-5.2 Logical Structure

The system structure uses the following logical elements.

- Cluster = { Node(s) }
  - A cluster is a collection of nodes that are logically connected. A node may belong to one or more clusters. A cluster may span subnets, networks or domains.

**Figure 1-5.2 Clusters**



**Cluster A** - Master/Slave Point-to-point Communication (i.e. Poll/Cyclic/COS)

- A Master and its Slaves are a cluster
- A Master may also be a slave to another Master

**Cluster B** - Multicast Master/Slave Communication (i.e. Strobe)

- A Master and its Slaves are a cluster
- A Master may also participate with a peer in any cluster

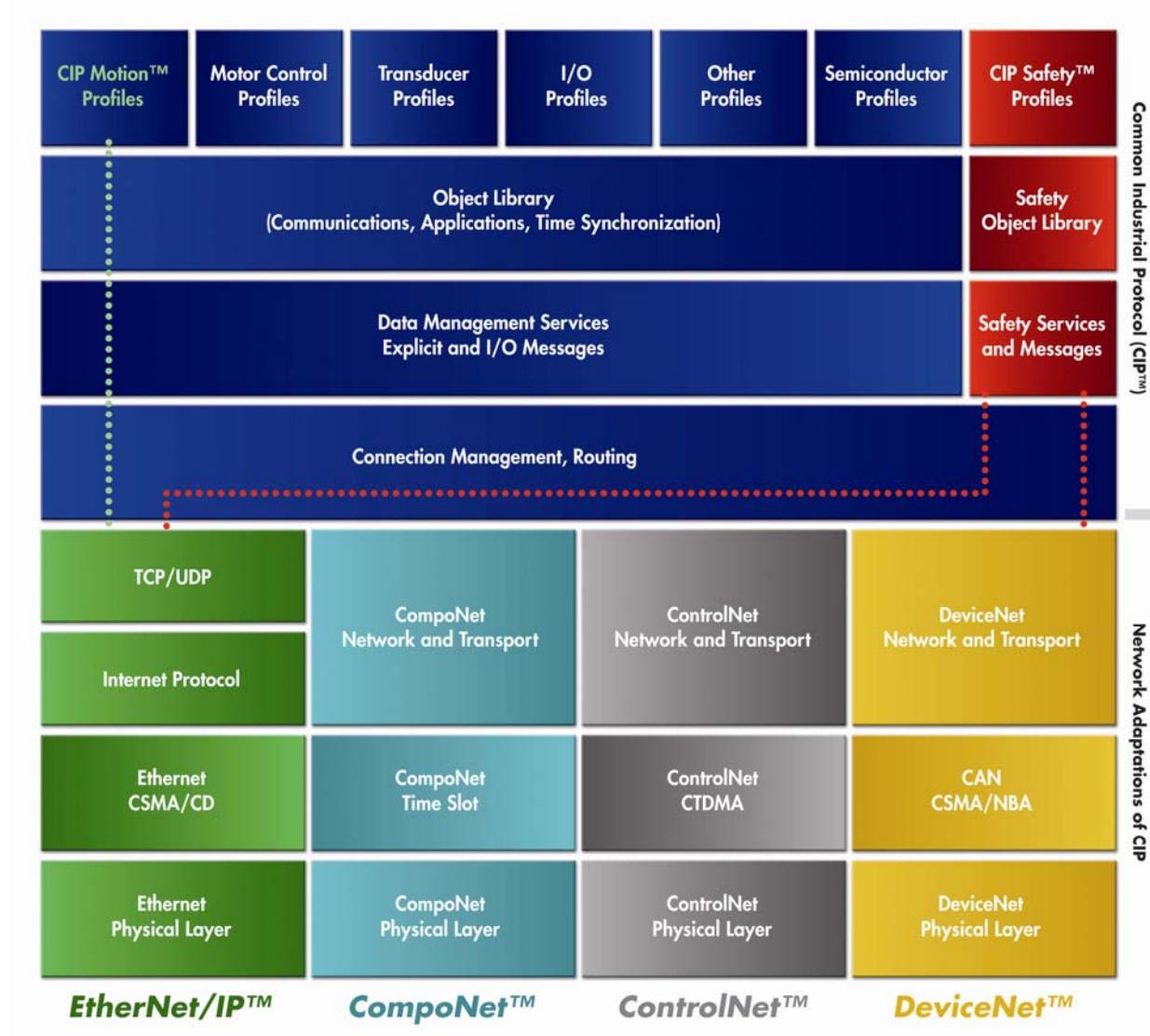
**Cluster C** - Peer-to-peer Communication (point-to-point or multicast)

- Nodes participating in a particular peer-to-peer relationship are a cluster

## 1-6 CIP Specification Structure

The CIP Networks Library is the definition of the application and user layers for a number of different network adaptations. The figure below shows the relationship between the specifications of this document and those of the CIP-based networks.

**Figure 1-6.1 CIP Architecture and Related Specifications**



## 1-7 Definitions

For the purposes of this standard, the following definitions apply.

Term	Definition
1-7.1 actual packet interval (API)	The measure of how frequently a specific connection produces its data.
1-7.2 allocate	To take a resource from a common area and assign that resource for the exclusive use of a specific entity.
1-7.3 application	Function or data structure for which data is consumed or produced.
1-7.4 application objects	Multiple object classes that manage and provide the run-time exchange of messages across the network and within the network device.
1-7.5 attribute	A description of an externally visible characteristic or feature of an object. The attributes of an object contain information about variable portions of an object. Typically, they provide status information or govern the operation of an object. Attributes may also affect the behavior of an object. Attributes are divided into class attributes and instance attributes.
1-7.6 behavior	Indication of how the object responds to particular events. Its description includes the relationship between attribute values and services.
1-7.7 big endian	A format for storage or transmission of binary data in which the most significant bit (or byte) comes first. The term comes from "Gulliver's Travels" by Jonathan Swift. The Lilliputians, being very small, had correspondingly small political problems. The Big-Endian and Little-Endian parties debated over whether soft-boiled eggs should be opened at the big end or the little end. See also: little-endian. [Source: RFC1392]
1-7.8 bit	A unit of information consisting of a 1 or a 0. This is the smallest data unit that can be transmitted.
1-7.9 byte	See octet.
1-7.10 class	A set of objects all of which represent a similar system component. A class is a generalization of the object, a template for defining variables and methods. All objects in a class are identical in form and behavior, but they may contain different attribute values.
1-7.11 class specific service	A service defined by a particular object class to perform a required function that is not performed by a common service. A class specific object is unique to the object class that defines it.
1-7.12 client	(1) An object that uses the services of another (server) object to perform a task. (2) An initiator of a message to which a server reacts.
1-7.13 communication objects	Components that manage and provide run-time exchange of messages across the network such as the Connection Manager object, the unconnected message manager (UCMM), and the Message Router object.
1-7.14 connection	A logical binding between two application objects. These application objects may be the same or different devices.
1-7.15 connection ID (CID)	Identifier assigned to a transmission that is associated with a particular connection between producers and consumers that identifies a specific piece of application information.
1-7.16 connection path	Is made up of a byte stream that defines the application object to which a connection instance applies.
1-7.17 consume	The act of receiving data from a producer.
1-7.18 consumer	A node that is receiving data from a producer.
1-7.19 consuming application	The application that consumes data.
1-7.20 cyclic	Term used to describe events that repeat in a regular and repetitive manner.
1-7.21 device	A physical hardware connection to the link. A device may contain more than one node.

<b>Term</b>	<b>Definition</b>
1-7.22 device profile	A collection of device-dependent information and functionality providing consistency between similar devices of the same device type.
1-7.23 end node	A producing or consuming node.
1-7.24 end point	One of the communicating entities involved in a connection.
1-7.25 error	A discrepancy between a computed, observed or measured value or condition and the specified or theoretically correct value or condition.
1-7.26 instance	The actual physical presentation of an object within a class. Identifies one of many objects within the same object class.
1-7.27 instantiated	An object that has been created in a device.
1-7.28 library element	A derived or standard data type, function, function block, program or resource in IEC 1131 - programmable controllers.
1-7.29 link	Collection of nodes with unique MAC IDs. Segments connected by repeaters make up a link; links connected by routers make up a network.
1-7.30 little endian	Describes a model of memory organization that stores the least significant byte at the lowest address. On the network medium, the lowest order byte is transferred first. The native CIP data types are sent in little endian order. See appendix C of the CIP Common specification for a detailed description of this encoding.
1-7.31 Message Router	The object within a node that distributes messaging requests to the appropriate application objects.
1-7.32 multicast	A packet with a special destination address, which multiple nodes on the network may be willing to receive. [Source: RFC1392]
1-7.33 multicast connection	A connection from one node to many. Multicast connections allow a single producer to be received by many consumer nodes.
1-7.34 network	A series of nodes connected by some type of communication medium. The connection paths between any pair of nodes can include repeaters, routers and gateways.
1-7.35 network address or node address	An identification value assigned to each node on the CIP network. This value distinguishes a node among all other nodes on the same link. The format is network specific.
1-7.36 node	A connection to a link that requires a single MAC ID.
1-7.37 object	(1) An abstract representation of a particular component within a product. Objects can be composed of any or all of the following components: a) data (information which changes with time); b) configuration (parameters for behavior); c) methods (things that can be done using data and configuration). (2) A collection of related data (in the form of variables) and methods (procedures) for operating on that data that have clearly defined interface and behavior.
1-7.38 object specific service	A service defined by a particular object class to perform a required function that is not performed by a common service. An object specific service is unique to the object class that defines it.
1-7.39 octet	An octet is 8 bits that indicates no particular data type.
1-7.40 originator	The client responsible for establishing a connection path to the target.
1-7.41 point-to-point connection	A connection that exists between two nodes only. Connections can be either point-to-point or multicast.
1-7.42 port	A CIP port is the abstraction for a physical network connection to a CIP device. A CIP device has one port for each network connection. Note: network specific definitions may include additional definitions of this term within the context of the network.
1-7.43 produce	Act of sending data to a consumer.

<b>Term</b>	<b>Definition</b>
1-7.44 producer	A node that is responsible for transmitting data.
1-7.45 redundant media	A system using more than one medium to help prevent communication failures.
1-7.46 requested packet interval (RPI)	The measure of how frequently the originating application requires the transmission of data from the target application.
1-7.47 scheduled network	<b>A network which utilizes a time-based scheduling mechanism that provides network devices with deterministic and predictable access to the medium while preventing network collisions.</b>
1-7.48 serial number	A unique 32-bit integer assigned by each manufacturer to every device. The number need only be unique with respect to the manufacturer.
1-7.49 server	An object that provides services to another (client) object.
1-7.50 service	Operation or function that an object performs upon request from another object.
1-7.51 target	The end-node to which a connection is established.
1-7.52 tool	An executable software program that interacts with the user to perform some function.
1-7.53 unconnected message manager (UCMM)	The component within a node that transmits and receives unconnected explicit messages and sends them directly to the Message Router object.

## **1-8 Abbreviations**

For the purposes of this standard, the following abbreviations apply.

<b>Abbreviation</b>	<b>Meaning</b>
1-8.1 API	Actual Packet Interval
1-8.2 ASCII	American Standard Code for Information Interchange
1-8.3 ASN.1	Abstract Syntax Notation
1-8.4 CI	ControlNet International, Ltd.
1-8.5 CIP	The Common Industrial Protocol defined by the CIP Common Specification. CIP includes both connected and unconnected messaging.
1-8.6 CID	Connection Identifier
1-8.7 DLL	Data Link Layer
1-8.8 EPR	Expected Packet Rate
1-8.9 ISO	International Standards Organization
1-8.10 MAC ID	Media Access Control Identifier
1-8.11 PDU	Protocol Data Unit
1-8.12 ODVA	Open DeviceNet Vendor Association
1-8.13 O->T	Originator to Target (used to describe packets that are sent from the originator to the target)
1-8.14 OSI	Open Systems Interconnection (see ISO 7498)
1-8.15 RPI	Requested Packet Interval
1-8.16 SDU	Service Data Unit
1-8.17 SEM	State Event Matrix
1-8.18 SEMI	Semiconductor Equipment Materials International
1-8.19 STD	State Transition Diagram, used to describe object behavior
1-8.20 T->O	Target to Originator (used to describe packets that are sent from the target to the originator)
1-8.21 UCMM	Unconnected Message Manager

## **Volume 1: Common Industrial Protocol (CIP<sup>TM</sup>) Specification**

### **Chapter 2: Messaging Protocol**

---

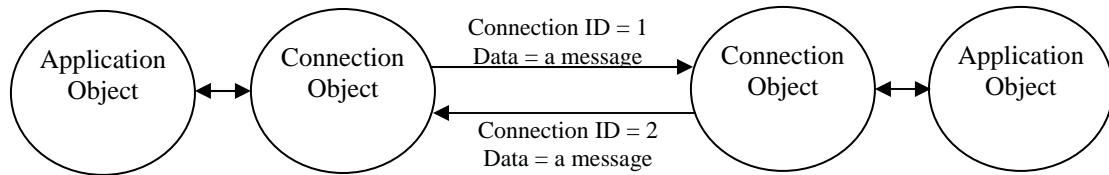
## Contents

2-1	Introduction.....	3
2-2	Connection Establishment Overview.....	4
2-2.1	Explicit Messaging and the UCMM .....	4
2-2.2	I/O Connections .....	7
2-3	Client and Server Connection Endpoints .....	9
2-4	Message Router Request/Response Formats .....	10

## 2-1 Introduction

CIP is layered on top of a *connection-based network*. A CIP *connection* provides a path between multiple applications. When a connection is established, the transmissions associated with that connection are assigned a **Connection ID (CID)**. If the connection involves a bi-directional exchange, then two Connection ID values are assigned. See Figure 2-1.1.

**Figure 2-1.1 Connections and Connection IDs**



The definition and format of the connection ID is network dependent. For example, the connection ID for CIP connections over DeviceNet is based on the CAN Identifier Field.

## **2-2 Connection Establishment Overview**

This section presents an overview of dynamically establishing both Explicit Messaging and I/O Connections.

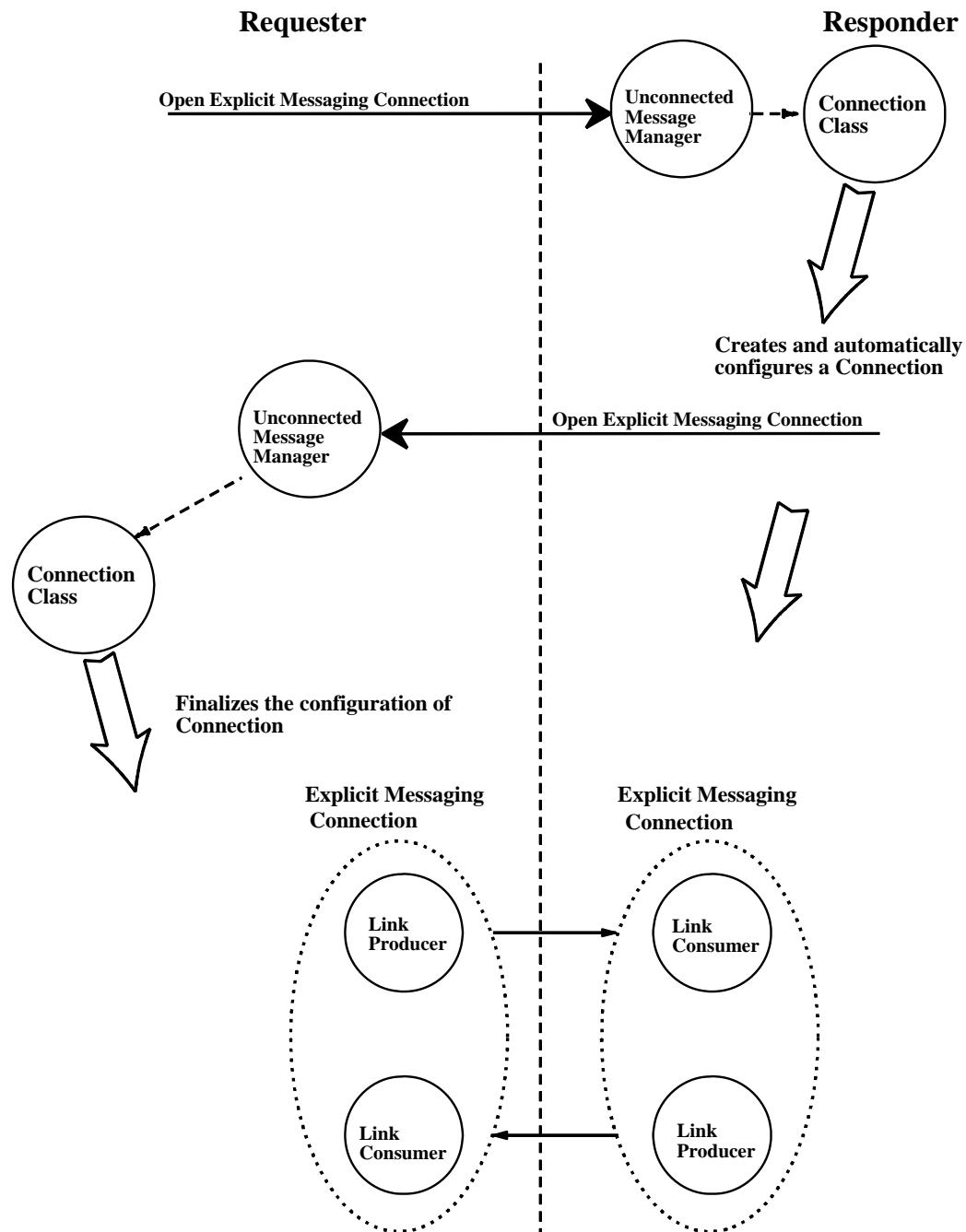
### **2-2.1 Explicit Messaging and the UCMM**

The Unconnected Message Manager (UCMM) is responsible for processing Unconnected Explicit Requests and Responses. This includes establishing both Explicit Messaging and I/O connections. The underlying network defines how the UCMM is accessed and may limit the messaging which can occur across the UCMM.

When using the UCMM to establish an explicit messaging connection, the target application object is the Message Router object (Class Code 2).

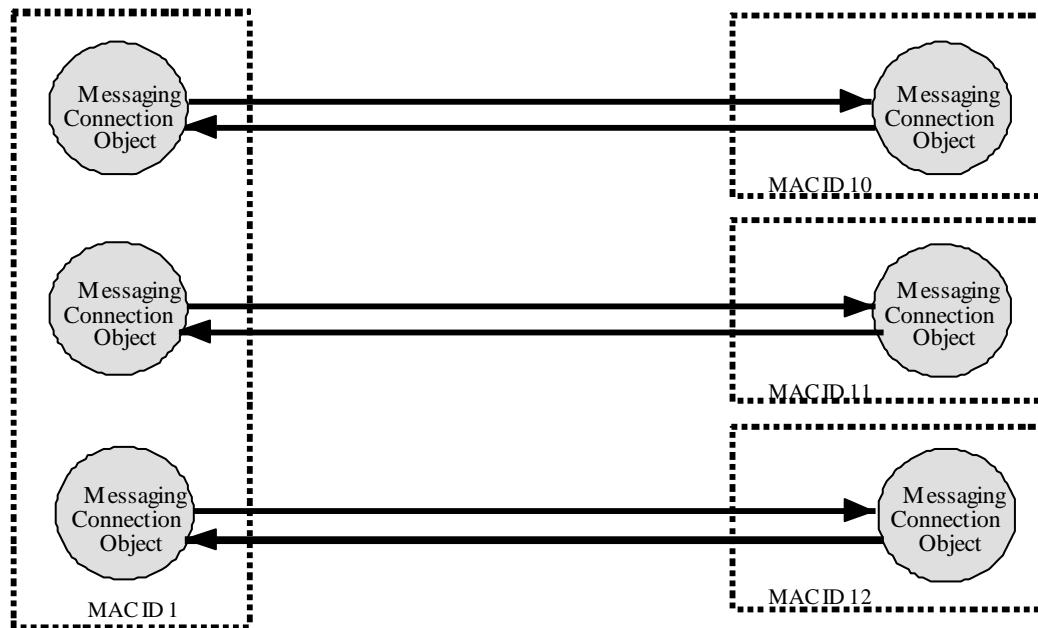
Figure 2-2.1 illustrates the steps involved in establishing a Messaging Connection.

Figure 2-2.1 Establishing an Explicit Messaging Connection



Explicit Messaging Connections are unconditionally **point-to-point**. Point-to-point connections exist between two devices ONLY. The device that requests that the connection be opened (the client) is one *end-point* of the connection, and the module that receives and responds to the request (the server) is the other *end-point*. See Figure 2-2.2.

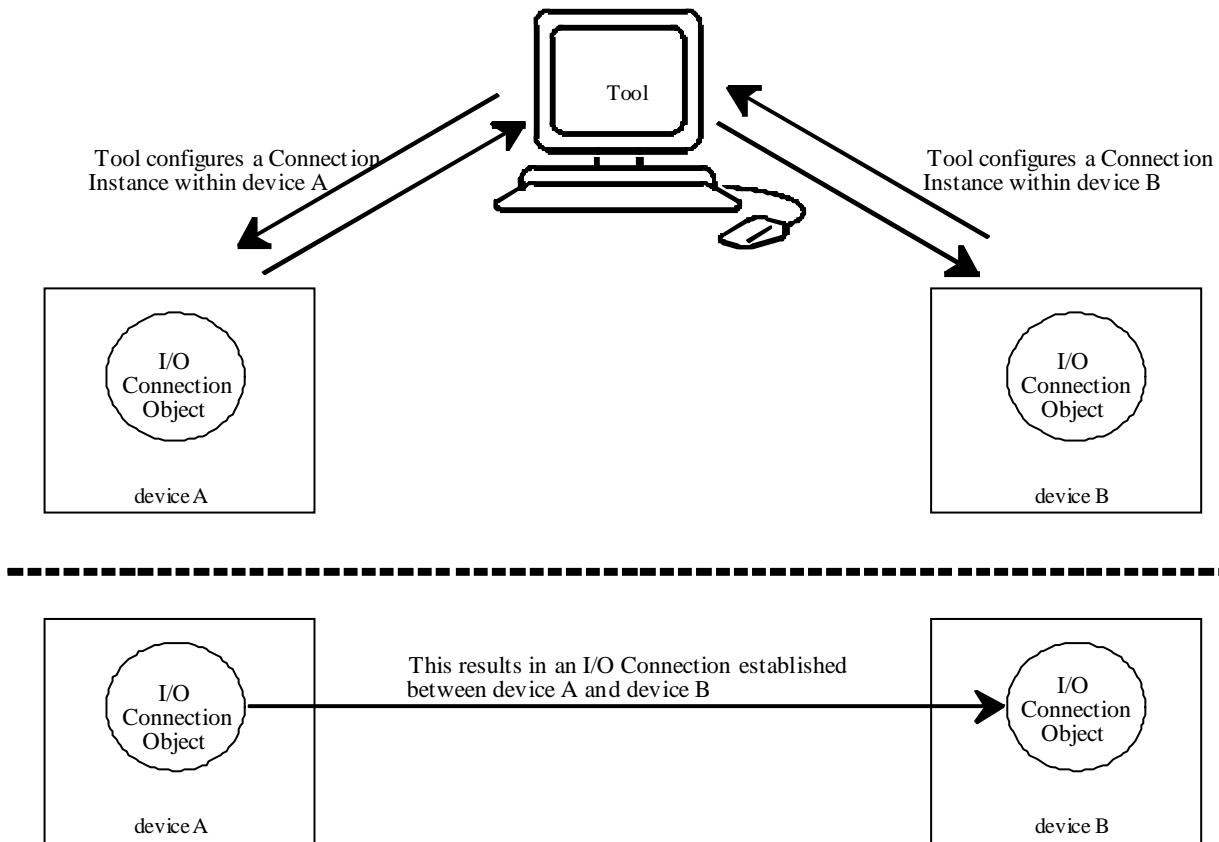
**Figure 2-2.2 Point-to-Point Nature of Explicit Messaging Connections**



## 2-2.2 I/O Connections

The CIP Network definition allows the establishment of a variety of I/O connections. This specification does not dictate any rules associated with *who* may perform Connection configuration. For example, a tool could interface with two separate devices and create an I/O Connection between them. See Figure 2-2.3.

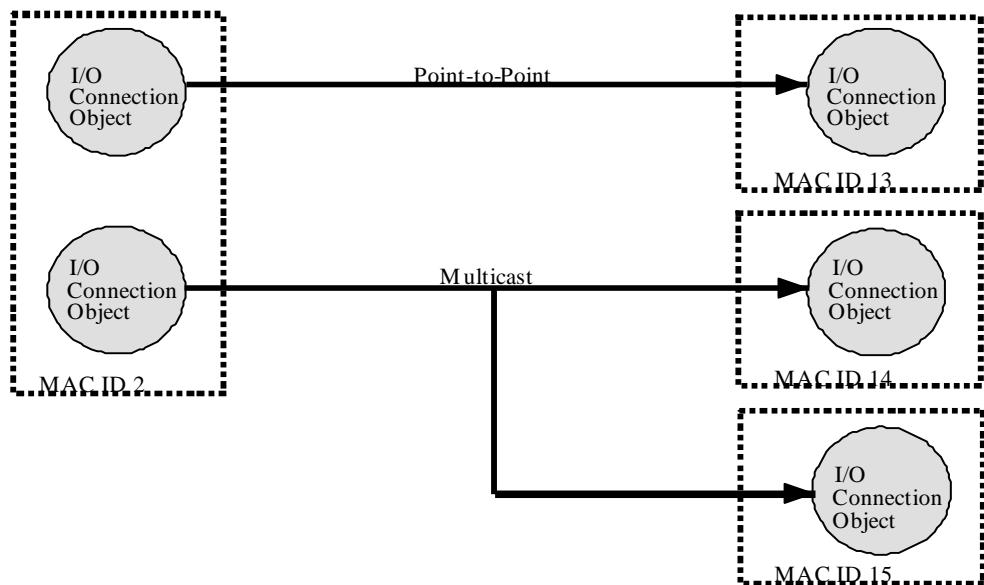
**Figure 2-2.3 Tool Interface with Devices to Create Connection**



The tool uses various Explicit Messaging Services to create and configure the I/O Connection Objects within the end points.

I/O connections can be either point-to-point or ***multicast***. Multicast connections allow a single transmission to be heard by many nodes. See Figure 2-2.4.

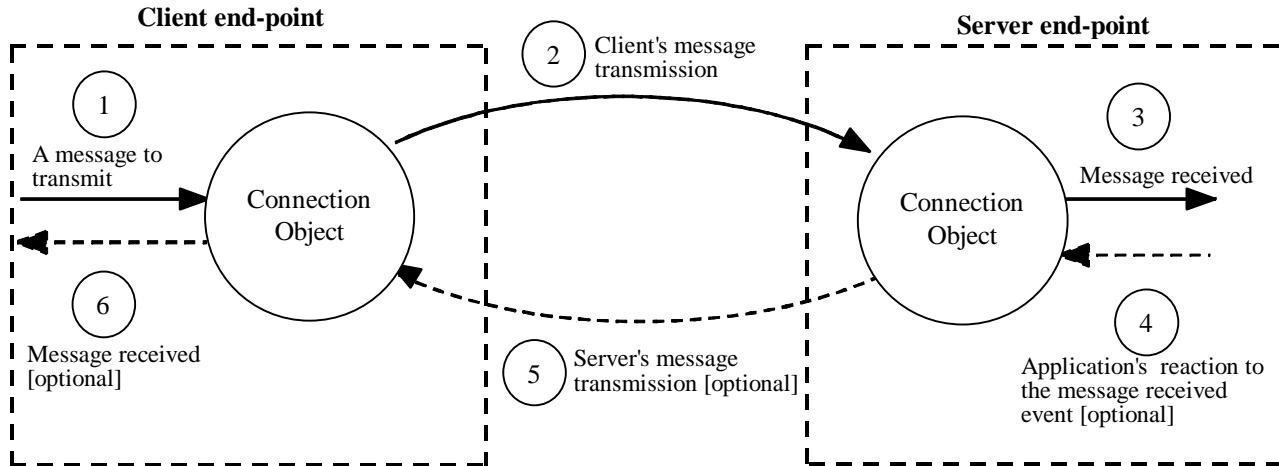
**Figure 2-2.4 Point-to-Point or Multicast Nature of I/O Connections**



## 2-3 Client and Server Connection Endpoints

The terms *Client* and *Server* are used throughout this document when discussing the behavior associated with a connection end-point. A Client end-point and Server end-point(s) are associated with both Explicit Messaging and I/O Connections. The *Client* is the module that originates a transmission, and the *Server* is the module that reacts to that transmission. The Server's reaction may cause it to return a message to the Client.

**Figure 2-3.1 Illustration of Client/Server Message Flow**



## 2-4 Message Router Request/Response Formats

CIP defines a standard data format for delivering data to and from the Message Router object. This data format is used in various places within CIP including the Unconnected Send service of the Connection Manager object and the UCMM data structures of most of the CIP networks.

The Message Router Request Format is defined in Table 2-4.1.

**Table 2-4.1 Message Router Request Format**

Parameter Name	Data Type	Description
Service	USINT	Service code of the request.
Request_Path_Size	USINT	The number of 16 bit words in the Request_Path field (next element).
Request_Path	Padded EPATH	This is an array of bytes whose contents convey the path of the request (Class ID, Instance ID, etc.) for this transaction.
Request_Data	Array of octet	Service specific data to be delivered in the Explicit Messaging Request. If no additional data is to be sent with the Explicit Messaging Request, then this array will be empty.

The Message Router Response Format is defined in Table 2-4.2.

**Table 2-4.2 Message Router Response Format**

Parameter Name	Data Type	Description
Reply Service	USINT	Reply service code.
Reserved	octet	Shall be zero.
General Status	USINT	One of the General Status codes listed in Appendix B (Status Codes).
Size of Additional Status	USINT	Number of 16 bit words in Additional Status array.
Additional Status	Array of WORD	Additional status.
Response Data	Array of octet	Response data from request or additional error data if General Status indicated an error.

## **Volume 1: Common Industrial Protocol (CIP<sup>TM</sup>) Specification**

### **Chapter 3: Communication Object Classes**

## Contents

3-1	Introduction.....	5
3-1.1	Creating Connections Through the Connection Object.....	5
3-1.2	Creating Connections Through the Connection Manager Object .....	5
3-2	Link Producer Object Class Definition .....	6
3-2.1	Link Producer Object Class Attributes.....	6
3-2.2	Link Producer Object Class Services .....	6
3-2.3	Link Producer Object Instance Attributes.....	6
3-2.3.1	state, USINT data type .....	6
3-2.3.2	connection_id .....	6
3-2.4	Link Producer Object Instance Services .....	6
3-2.5	Link Producer Instance Behavior.....	7
3-3	Link Consumer Object Class Definition .....	8
3-3.1	Link Consumer Object Class Attributes.....	8
3-3.2	Link Consumer Class Services.....	8
3-3.3	Link Consumer Instance Attributes.....	8
3-3.3.1	state, USINT data type .....	8
3-3.3.2	connection_id .....	8
3-3.4	Link Consumer Instance Services .....	8
3-3.5	Link Consumer Instance Behavior.....	9
3-4	Connection Object Class Definition.....	10
3-4.1	Connection Object Class Attributes .....	10
3-4.2	Connection Object Class Services.....	11
3-4.3	Connection Class Object Specific Services.....	11
3-4.3.1	Connection Bind Service.....	11
3-4.3.2	Producing Application Lookup Service .....	12
3-4.4	Connection Object Instance Attributes .....	13
3-4.4.1	State, Attribute 1 - USINT data type.....	14
3-4.4.2	Instance_type, Attribute 2 - USINT data type.....	15
3-4.4.3	TransportClass_trigger, Attribute 3 - BYTE data type.....	15
3-4.4.3.1	Server Transport Class 0 and 1 Behavior.....	18
3-4.4.3.2	Server Transport Class 2 Behavior .....	19
3-4.4.3.3	Server Transport Class 3 Behavior .....	20
3-4.4.3.4	Client Transport Class 0 and 1 Behavior: Cyclic .....	21
3-4.4.3.5	Client Transport Class 2 and 3 Behavior: Cyclic .....	22
3-4.4.3.6	Client Transport Class 0 and 1 Behavior: Change-Of-State .....	23
3-4.4.3.7	Client Transport Class 2 and 3 Behavior: Change-Of-State .....	24
3-4.4.3.8	Client Transport Class 0 and 1 Behavior: Application Object Triggered .....	25
3-4.4.3.9	Client Transport Class 2 and 3 Behavior: Application Object Triggered .....	26
3-4.4.4	DeviceNet_produced_connection_id, Attribute 4 – UINT data type .....	27
3-4.4.5	DeviceNet_consumed_connection_id, Attribute 5 – UINT data type .....	27
3-4.4.6	DeviceNet_initial_comm_characteristics, Attribute 6 – USINT data type .....	27
3-4.4.7	Produced_connection_size, Attribute 7 – UINT data type.....	28
3-4.4.8	Consumed_connection_size, Attribute 8 – UINT data type .....	28
3-4.4.9	Expected_packet_rate, Attribute 9 – UINT data type .....	29
3-4.4.10	CIP_produced_connection_id, Attribute 10 – UDINT data type .....	30
3-4.4.11	CIP_consumed_connection_id, Attribute 11 – UDINT data type .....	30
3-4.4.12	Watchdog_timeout_action, Attribute 12 – USINT data type .....	30
3-4.4.13	Produced_connection_path_length, Attribute 13 – UINT data type .....	30
3-4.4.14	Produced_connection_path, Attribute 14 – EPATH data type.....	31
3-4.4.15	Consumed_connection_path_length, Attribute 15 – UINT data type .....	31
3-4.4.16	Consumed_connection_path, Attribute 16 – EPATH data type.....	31
3-4.4.17	Production_inhibit_time, Attribute 17 – UINT data type.....	31
3-4.4.18	Connection_timeout_multiplier, Attribute 18 –USINT data type .....	32

3-4.4.19	Connection_binding_list, Attribute 19 – Struct(UINT(size), Array of UINT[size]) data type .....	32
3-4.5	Connection Timing .....	32
3-4.5.1	Transmission Trigger Timer.....	32
3-4.5.2	Inactivity/Watchdog Timer .....	33
3-4.5.3	Production Inhibit Timer.....	35
3-4.6	Connection Object Instance Services.....	36
3-4.7	Connection Instance Behavior .....	39
3-4.7.1	I/O Connection Instance Behavior .....	39
3-4.7.2	CIP Bridged Connection Instance Behavior .....	45
3-4.7.3	Explicit Messaging Connection Instance Behavior.....	47
3-4.8	Connection Object Attribute Access Rules.....	50
3-5	Connection Manager Object Class Definition .....	53
3-5.1	Connection Manager Object Class Attributes .....	53
3-5.2	Connection Manager Object Class Services .....	53
3-5.2.1	Class Level Get_Attributes_All Response .....	53
3-5.3	Connection Manager Object Instance Attributes .....	54
3-5.4	Connection Manager Object Instance Common Services .....	55
3-5.4.1	Instance Level Get_Attributes_All Response .....	55
3-5.5	Connection Manager Object Instance Object Specific Services .....	56
3-5.5.1	Connection Manager Object Specific Service Parameters .....	57
3-5.5.1.1	Network Connection Parameters.....	57
3-5.5.1.2	Packet Interval .....	58
3-5.5.1.3	Connection Timing .....	59
3-5.5.1.4	Connection Serial Number.....	61
3-5.5.1.5	Connection Timeout Multiplier.....	61
3-5.5.1.6	Vendor ID .....	61
3-5.5.1.7	Originator Serial Number.....	61
3-5.5.1.8	Connection Number .....	61
3-5.5.1.9	Connection Path Size .....	62
3-5.5.1.10	Connection Path .....	62
3-5.5.1.11	Network Connection ID .....	66
3-5.5.1.12	Transport Class and Trigger.....	66
3-5.5.2	Forward_Open and Large_Forward_Open .....	66
3-5.5.3	Forward_Close .....	69
3-5.5.4	Unconnected_Send.....	70
3-5.5.5	Get_Connection_Data .....	73
3-5.5.6	Search_Connection_Data.....	74
3-5.6	Connection Manager Object Instance Error Codes .....	74
3-5.6.1	Error Code Listing.....	74
3-6	Application Connection Type using Class 0 or 1 Transports.....	81
3-6.1	Real time formats including RUN/IDLE notification .....	82
3-6.1.1	Modeless Format.....	82
3-6.1.2	Zero Length Data Format.....	82
3-6.1.3	Heartbeat Format.....	83
3-6.1.4	32-Bit Header Format .....	83
3-6.2	Configuration .....	83
3-6.3	Specifying Different Application Connection Types .....	84
3-6.4	Application types .....	84
3-6.4.1	Listen only.....	84
3-6.4.2	Input only .....	84
3-6.4.3	Exclusive Owner .....	84
3-6.4.4	Redundant owner .....	85
3-6.4.4.1	General.....	85
3-6.4.4.2	Establishing the Connection.....	86
3-6.4.4.3	Redundant owner O->T data format .....	86
3-6.4.4.4	Determining the owning connection .....	87

3-6.4.4.5	Transporting events and data to a target application.....	87
3-7	Port Object Class Definition .....	88
3-7.1	Port Object Class Attributes.....	88
3-7.2	Port Object Class Services .....	89
3-7.3	Port Object Instance Attributes .....	89
3-7.4	Port Object Instance Common Services.....	91
3-7.4.1	Get_Attributes_All Response.....	91

## **3-1 Introduction**

The CIP Communication Objects manage and provide the run-time exchange of messages. The Services, Attributes, and Behaviors associated with the Communication Objects are detailed in this Chapter. Part of an object definition involves assigning a Data Type to an attribute. See Appendix C for a detailed description of CIP Data Types and Data Management.

**Important:** It is not the intent of the following sections to specify any particular internal implementation.

The Communication Object Classes are defined by describing:

- Object Class Attributes
- Object Class Services
- Object Instance Attributes
- Object Instance Services
- Object Instance Behavior

Each CIP connection is represented by a Connection Object (Class code 0x05). The creation of this communication object resource can be done in one of two ways. Each subnet type defines which method shall be used. The two methods are:

- Use of the Create service (Service code 0x08) for the Connection Object
- Use of the Forward Open service for the Connection Manager Object

Subnet types may define other methods of creating this communication object resource. An example of this is DeviceNet's Allocate\_Master/Slave\_Connection\_Set service. See the appropriate specification volume for details of subnet type-specific methods.

### **3-1.1 Creating Connections Through the Connection Object**

When the subnet defines that connections are created through the Connection Object, a CIP device shall support the Create service for this class. The Create service instantiates a Connection Instance with attribute values defaulted as defined by the class. The connection instance is configured through individual access to each Connection instance attribute. A separate service request (Apply\_Attributes, Service code 0x0D) is needed to transition the connection to the Established state.

### **3-1.2 Creating Connections Through the Connection Manager Object**

When the subnet defines that connections are created through the Connection Manager Object, a CIP device shall support the Forward Open service of this class. When successful, the Connection Manager instantiates an instance of the Connection class. This connection instance is configured with the values sent in the Forward Open service and is transitioned to the established state. This single CIP service request is modeled internally as a single Connection Class service request (using the Create service) and several internal service requests (using the Set\_Attribute\_Single and Apply\_Attributes services). A device supporting connection creation through the Connection Manager may or may not provide external visibility to the Connection Class instances.

## **3-2 Link Producer Object Class Definition**

The Link Producer Object is the component responsible for the low-level transmission of data.

**Important:** NO externally visible interface to the Link Producer Class across Explicit Messaging Connections exists. All services/attributes noted in the following sections describe internal behavior. These services/attributes are accessed via attributes and services of the Connection Object.

### **3-2.1 Link Producer Object Class Attributes**

There are no Link Producer Class Attributes.

### **3-2.2 Link Producer Object Class Services**

The services supported by the Link Producer Class are listed below.

- Create - Used internally to instantiate a Link Producer Object
- Delete - Used internally to delete a Link Producer Object

### **3-2.3 Link Producer Object Instance Attributes**

The following list describes the Link Producer Instance attributes.

#### **3-2.3.1 state, USINT data type**

This contains the current state of the Link Producer instance. Possible states include the following:

**Table 3-2.1 Link Producer States**

State Name	Description
Non-existent	The Link Producer has yet to be instantiated
Running	The Link Producer has been instantiated and is waiting to be told to transmit via the invocation of its Send service.

#### **3-2.3.2 connection\_id**

The value placed within the message frame when this Link Producer is triggered to send. The placement and format of this value within the message frame is subnet type-specific. The Connection Object using this Link Producer internally initializes this attribute with the value in its **produced\_connection\_id** attribute. See section 3-4.4, Connection Instance Attributes for a definition of the produced\_connection\_id attributes.

### **3-2.4 Link Producer Object Instance Services**

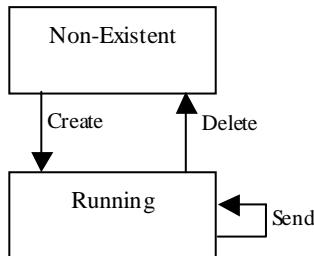
The services supported by a Link Producer Object Instance are listed below.

- Send - Used internally to tell the Link Producer to transmit data onto the subnet.
- Get\_Attribute - Used internally to read a Link Producer Object attribute
- Set\_Attribute - Used internally to modify a Link Producer Object attribute

### 3-2.5 Link Producer Instance Behavior

Figure 3-2.1 and Table 3-3.2 illustrate the Link Producer's Instance behavior.

**Figure 3-2.1 Link Producer in Object State Transition**



**Table 3-2.2 State/Event Matrix: Link Producer**

Event	State	
	Non-Existent	Running
Class Create invoked internally	Class instantiates Link Producer Object. Link Producer enters the Running state.	Not applicable
Class Delete invoked internally	Error: Instance Not Present	Release all associated instance resources. Transition to the Non-existent state.
Send invoked internally	Error: Instance Not Present	Transmit the data
Set_Attribute invoked internally	Error: Instance Not Present	Modify the attribute
Get_Attribute invoked internally	Error: Instance Not Present	Return the attribute value

## **3-3 Link Consumer Object Class Definition**

The Link Consumer Object is the component responsible for the low-level reception of messages.

**Important:** NO externally visible interface to the Link Consumer Class across Explicit Messaging Connections exists. All services/attributes noted in the following sections describe internal behavior. These services/attributes are accessed via attributes and services of the Connection Object.

### **3-3.1 Link Consumer Object Class Attributes**

There are no Link Consumer Class Attributes.

### **3-3.2 Link Consumer Class Services**

The services supported by the Link Consumer Class are listed below.

- Create - Used internally to instantiate a Link Consumer Object
- Delete - Used internally to delete a Link Consumer Object

### **3-3.3 Link Consumer Instance Attributes**

The following list describes the Link Consumer Instance attributes.

#### **3-3.3.1 state, USINT data type**

This contains the current state of the consumer instance. Possible states include:

**Table 3-3.1 Link Consumer States**

State Name	Description
Non-existent	The Link Consumer has yet to be instantiated.
Running	The Link Consumer has been instantiated and is waiting to receive a message.

#### **3-3.3.2 connection\_id**

This attribute holds the Connection ID value in the message frame that denotes the message to be received by this consumer. The placement and format of this value within the message frame is subnet type-specific. The Connection Object utilizing this Link Consumer internally initializes this attribute with the value in its **consumed\_connection\_id** attribute. See section 3-4.4, Connection Instance Attributes for a definition of the consumed\_connection\_id attributes.

### **3-3.4 Link Consumer Instance Services**

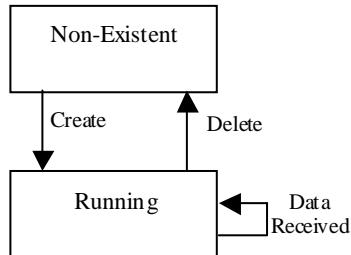
The services supported by a Link Consumer Object Instance are listed below.

- Get\_Attribute - Used internally to read a Link Consumer Object attribute
- Set\_Attribute - Used internally to modify a Link Consumer Object attribute

### 3-3.5 Link Consumer Instance Behavior

Figure 3-3.1 and Table 3-3.2 illustrate the Link Consumer's Instance behavior.

**Figure 3-3.1 Link Consumer Object State Transition Diagram**



**Table 3-3.2 State/Event Matrix: Link Consumer**

Event	State	
	Non-Existent	Running
Class Create invoked internally	Class instantiates Link Consumer Object. Link Consumer enters the Running state.	Not applicable
Class Delete invoked internally	Error: Instance Not Present	Release all associated instance resources. Transition to the Non-existent state.
Data received	Not applicable	Deliver data to the associated Connection Object by invoking its Receive_Data() service
Set_Attribute invoked internally	Error: Instance Not Present	Modify the attribute
Get_Attribute invoked internally	Error: Instance Not Present	Return the attribute value

## 3-4 Connection Object Class Definition

### Class Code: 05 hex

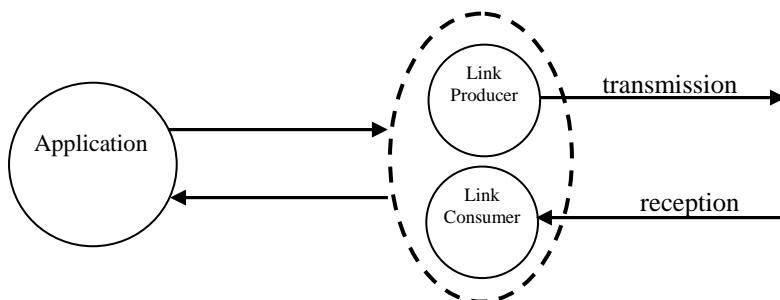
The Connection Class allocates and manages the internal resources associated with both I/O and Explicit Messaging Connections. The specific instance generated by the Connection Class is referred to as a *Connection Instance* or a *Connection Object*.

Unless otherwise noted, all services/attributes noted in the following sections are accessible using Explicit Messaging.

A Connection Object within a particular module actually represents one of the end-points of a Connection. It is possible for one of the Connection end-points to be configured and “active” (e.g. transmitting) without the other end-point(s) being present. Connection Objects are used to model the communication specific characteristics of a particular Application-to-Application(s) relationship.

A specific Connection Object Instance manages the communication-specific aspects related to an end-point. A CIP Connection Object uses the services provided by a Link Producer and/or Link Consumer to perform low-level data transmission and reception functions.

**Figure 3-4.1 Connection Object & Link Producer/Consumer Relationship**



### 3-4.1 Connection Object Class Attributes

The Connection Class attributes are defined below in Table 3-4.1.

**Table 3-4.1 Connection Class Attributes**

Attribute ID	Need In Implementation	Access Rule	Attribute Name	Data Type	Attribute Description	Semantics of Values
1 thru 7	These class attributes are optional and are described in Chapter 4 of this specification.					
8	Conditional <sup>1</sup>		Connection Request Error Count		See CIP Safety Specification (Volume 5, Chapter 5)	
9	Conditional <sup>1</sup>		Safety Connection Counters			

<sup>1</sup> These attributes are not allowed if the device is not a safety device. If the device is a safety device, see Volume 5.

### 3-4.2 Connection Object Class Services

The Connection Class supports the following CIP Common Services:

**Table 3-4.2 Connection Class Services**

Service Code	Need In Implementation	Service Name	Service Description
08hex	Optional	Create	Used to instantiate a Connection Object
09hex	Optional	Delete	Used to delete all Connection Objects (independent of state) and to release all associated resources. When the Delete service is sent to the Connection Class (Instance ID set to zero (0)) versus a specific Connection Object Instance, then ALL Instances are deleted.
05hex	Optional	Reset	Used to reset all resettable Connection Objects. The conditions under which a Connection is resettable are listed in the State Event Matrix in section 3-4.7
11hex	Optional	Find_Next_Object_Instance	Used to search for Instance IDs associated with existing Connection Objects. The Connection Class returns the Instance ID associated with any Connection Object not in the <b>Non-Existent</b> state.
0Ehex	Conditional	Get_Attribute_Single	Used to read a Connection Class attribute value. This service is Required if any of the Connection Class Attributes are supported.

### 3-4.3 Connection Class Object Specific Services

The following class specific services are defined for the connection object class.

**Table 3-4.3 Connection Class Object Specific Services**

Service Code (hex)	Need In Implementation	Service Name	Service Description
4Bhex	Conditional <sup>1</sup>	Connection Bind	Binds two connection instances
4Chex	Conditional <sup>1</sup>	Producing Application Lookup	Find established connections that are producing data from the application object specified by this service
4Ehex	Conditional <sup>2</sup>	SafetyClose	See CIP Safety Specification
54hex	Conditional <sup>2</sup>	SafetyOpen	(Volume 5, Chapter 3)

1 This service is required if the Create service is supported.

2 These services are not allowed if the device is not a safety device. If the device is a safety device, see Volume 5

#### 3-4.3.1 Connection Bind Service

The Connection Bind object specific service binds two dynamically created I/O connection instances together for purposes of connection timeouts and deletions. A dynamically created I/O connection is the result of a Create service to the Connection class with the Instance\_type attribute set to I/O (attribute value of 1). If one of these I/O connection instance times out or is deleted, the other instance shall exhibit the same behavior.

The Connection Bind service has the following request parameter:

**Table 3-4.4 Connection Bind Service Request Parameter**

Parameter Name	Data Type	Parameter Description
Bound Instances	STRUCT of UINT UINT	Instance numbers of connection objects to be bound.

There is no response data. The service request may return a status from the following list of status codes:

**Table 3-4.5 Connection Bind Service Status Codes**

General Status Code	Extended Status Code	Status Description
0x02	0x01	One or both of the connection instances is non-existent.
0x02	0x02	The connection class and/or instance is out of resources to bind instances.
0x0C	0x01	Both of the connection instances are existent, but at least one is not in the Established state.
0x20	0x01	Both connection instances are the same value.
0xD0	0x01	One or both of the connection instances is not a dynamically created I/O connection.
0xD0	0x02	One or both of the connection instances were created internally and the device is not allowing a binding to it.

### 3-4.3.2 Producing Application Lookup Service

The Producing Application Lookup object specific service provides a mechanism to find one or more connection instances in the Established state producing data from a given application object. If the requested producing application path is being produced by any connection in the Established state, the instance number of each connection producing from that path is returned.

The Producing Application Lookup service has the following service request parameter:

**Table 3-4.6 Producing Application Lookup Service Request Parameter**

Parameter Name	Data Type	Parameter Description
Producing Application Path	EPATH	Connection path of producing application to be searched for within connections in the Established state. The path shall be a single Logical or Symbolic EPATH.

The Producing Application Lookup service has the following service response parameters:

**Table 3-4.7 Producing Application Lookup Service Response Parameter**

Parameter Name	Data Type	Parameter Description
Instance Count	UINT	Number of Instances returned in the Connection Instance List parameter.
Connection Instance List	Struct of UINT	List of instance numbers of connection producing the data from the requested connection path within the node.

The service request may return a status from the following list of status codes:

**Table 3-4.8 Producing Application Lookup Service Status Codes**

General Status Code	Extended Status Code	Status Description
0x02	0x01	The connection path was not found in any connection instance in the Established state.

**3-4.4 Connection Object Instance Attributes**

The following list provides a summary of the Connection Instance attributes and their associated data types.

**Table 3-4.9 Connection Object Instance Attributes**

Attr ID	Need In Implementation	Attribute Name	Data Type	Brief Description of Attribute
1	Required	State	USINT	State of the object
2	Required	Instance_type	USINT	Indicates either I/O or Messaging Connection
3	Required	TransportClass_trigger	BYTE	Defines behavior of the Connection
4	Conditional	DeviceNet_produced_connection_id	UINT	Placed in CAN Identifier Field when the Connection transmits on a DeviceNet subnet. Described in Vol 3, DeviceNet Adapation of CIP.
5	Conditional	DeviceNet_consumed_connection_id	UINT	CAN Identifier Field value that denotes message to be received on a DeviceNet subnet. Described in Vol 3, DeviceNet Adaptation of CIP.
6	Conditional	DeviceNet_initial_comm_characteristics	BYTE	Defines the Message Group(s) across which productions and consumptions associated with this Connection occur on a DeviceNet subnet. Described in Vol 3, DeviceNet Adaptation of CIP.
7	Required	Produced_connection_size	UINT	Maximum number of bytes transmitted across this Connection
8	Required	Consumed_connection_size	UINT	Maximum number of bytes received across this Connection
9	Required	Expected_packet_rate	UINT	Defines timing associated with this Connection
10	Conditional	CIP_produced_connection_id	UDINT	Identifies the message sent on the subnet by this connection.
11	Conditional	CIP_consumed_connection_id	UDINT	Identifies the message received from the subnet for this connection.
12	Required	Watchdog_timeout_action	USINT	Defines how to handle Inactivity/Watchdog timeouts
13	Required	Produced_connection_path_length	UINT	Number of bytes in the produced_connection_path attribute
14	Required	Produced_connection_path	Packed EPATH	Specifies the Application Object(s) whose data is to be produced by this Connection Object. See Appendix C.
15	Required	Consumed_connection_path_length	UINT	Number of bytes in the consumed_connection_path attribute
16	Required	Consumed_connection_path	Packed EPATH	Specifies the Application Object(s) that are to receive the data consumed by this Connection Object. See Appendix C.

Attr ID	Need In Implementation	Attribute Name	Data Type	Brief Description of Attribute
17	Conditional	Production_inhibit_time	UINT	Defines minimum time between new data production. This attribute is required for all I/O Client connections, except those with a production trigger of Cyclic.
18	Optional	Connection_timeout_multiplier	USINT	Specifies the multiplier applied to the expected_packet_rate value to derive the value for the Inactivity/Watchdog Timer.
19	Conditional	Connection_binding_list	STRUCT UINT Array of UINT	List of I/O connection instances bound to this instance.

**Important:** Access Rules for the Connection Object Instance Attributes are defined in section 3-4.8.

The list below provides detailed descriptions of the Connection Object Instance Attributes. The defined **default** attribute values are to be used when no other internal and/or system-defined rules exist.

#### 3-4.4.1 State, Attribute 1 - USINT data type

This attribute defines the current state of the Connection instance. Table 3-4.10 defines the possible states and assigns a value used to indicate that state. Also see Figure 3-4.17 and Figure 3-4.20 for state transition behavior.

**Table 3-4.10 Values assigned to the state attribute**

Value	State Name	Description
00	Non-existent	The Connection has yet to be instantiated
01	Configuring	The Connection has been instantiated and is waiting for the following events to occur: (1) to be properly configured and (2) to be told to apply the configuration.
02	Waiting For Connection ID <sup>2</sup>	The Connection instance is waiting exclusively for its consumed_connection_id and/or produced_connection_id attribute to be set. <sup>1</sup>
03	Established	The Connection has been validly/fully configured and the configuration has been successfully applied.
04	Timed Out	If a Connection Object experiences an Inactivity/Watchdog timeout, then a transition <u>may</u> be made to this state. See the watchdog_timeout_action attribute description and the description of the Inactivity/Watchdog Timer (section 3-4.5) for more details.
05	Deferred Delete <sup>2</sup>	If an Explicit Messaging Connection Object experiences an Inactivity/Watchdog timeout, then a transition <u>may</u> be made to this state. See the watchdog_timeout_action attribute description and the description of the Inactivity/Watchdog Timer (section 3-4.5) for more details.
06	Closing	A CIP Bridged connection object has received, and is processing, a Forward Close from the Connection Manager. The deletion of the connection does not occur until after a successful Forward Open response has been received from the target node.

1 When the Connection instance attributes are applied it may not be possible for the module to generate the produced\_connection\_id and/or consumed\_connection\_id values (see initial\_comm\_characteristics attribute for rules). If this is the case then all the tasks required to apply the attributes are performed except for the initialization of these attributes within the Connection and associated Link Producer/Consumer Objects and a transition is made to this state. In this state the Connection instance is waiting exclusively for its produced\_connection\_id and/or consumed\_connection\_id attributes to be set.

2 This value is only used on DeviceNet.

**Important:** A dynamically created connection instance is the child of the Explicit Messaging connection across which it was created. Different subnet types may provide other mechanisms for creating a connection that imply a particular parent-child relationship. See network-specific specifications for details.

**Important:** All resources associated with a Connection Instance (*A*) that has been dynamically created across an Explicit Messaging Connection (*B*) must be released if the Explicit Messaging Connection (*B*) times out prior to the dynamically created Connection Instance (*A*) transitioning to the Established state.

**Important:** When a transition is made to the Established state, all timers associated with the Connection Object are activated (see section 3-4.5, Connection Timing)

#### **3-4.4.2 Instance\_type, Attribute 2 - USINT data type**

This attribute defines the instance type. See Table 3-4.11.

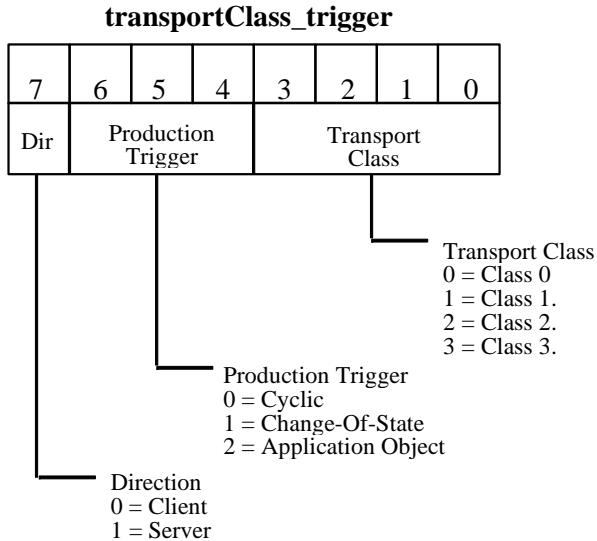
**Table 3-4.11 Values assigned to the instance\_type attribute**

Value	Meaning
00	Explicit Messaging. This Connection Instance represents one of the end-points of an Explicit Messaging Connection. An Explicit Messaging Connection is dynamically created by sending the <i>Open Explicit Messaging Connection Request</i> to the Connection Class.
01	I/O. This Connection Instance represents one of the end-points of an I/O Connection. An I/O Connection is dynamically created by sending a <i>Create Request</i> to the Connection Class.
02	CIP Bridged. This Connection Instance represents an intermediate ‘hop’ of a bridged I/O or Explicit Messaging connection. A pair of CIP Bridged connection objects (one for each subnet bridged between) are dynamically created by a node after successfully receiving a Forward Open service to the Connection Manager object when this node is not the end point (target). See Chapter 10 for additional information.

#### **3-4.4.3 TransportClass\_trigger, Attribute 3 - BYTE data type**

Defines whether this is a producing only, consuming only, or both producing and consuming connection. If this end point is to perform a data production, this attribute also defines the event that triggers the production. The eight (8) bits are divided as follows:

Figure 3-4.2 Transport Class Trigger Attribute



The **Direction bit** of the **transportClass\_trigger** byte indicates whether the end-point is to act as the Client or the Server on this connection. The following values are defined:

Table 3-4.12 Possible Values within Direction Bit

Value	Meaning	
0	Client	This end-point provides the Client behavior associated with this Connection. Additionally, this value indicates that the <i>Production Trigger</i> bits within the <b>transportClass_trigger</b> byte contain the description of <i>when</i> the Client is to produce the message associated with this connection. Client connections with production trigger value of 0 or 1 (Cyclic or Change-of-State) shall produce immediately after transitioning to the Established state.
1	Server	This end-point provides the Server behavior associated with this Connection. In addition, this value indicates that the <i>Production Trigger</i> bits within the <b>transportClass_trigger</b> byte are to be IGNORED. The <i>Production Trigger</i> bits are ignored due to the fact that a Server end-point <i>reacts</i> to the transmission from the Client. The only means by which a Server end-point is triggered to transmit is when this <i>reaction</i> calls for the production of a message (Transport Classes 2 or 3).

The following table lists the values that are possible within the **Production Trigger** bits of the **transportClass\_trigger** attribute.

**Table 3-4.13 Possible Values within Production Trigger Bits**

<b>If the value is:</b>	<b>Then the Production of a message is:</b>	
0	Cyclic	The expiration of the Transmission Trigger Timer triggers the data production. See section 3-4.5, Connection Timing for a detailed description of the Transmission Trigger Timer.
1	Change-Of-State	Production occurs when a change-of-state is detected by the Application Object. Note that the consuming end-point may have been configured to expect the packet at a certain rate, regardless of the triggering mechanism at the producing end-point. See the description of the <code>expected_packet_rate</code> attribute of a Connection Object and the description of Connection Timing in section 3-4.5 for more information.
2	Application Object Triggered	The Application Object decides when to trigger the production. Note that the consuming end-point may have been configured to expect the packet at a certain rate, regardless of the triggering mechanism at the producing end-point. See the description of the <code>expected_packet_rate</code> attribute of a Connection Object and the description of Connection Timing in section 3-4.5 for more information.
3 - 7	Reserved by CIP	

Table 3-4.14 lists possible values within the ***Transport Class*** nibble of the **`transportClass_trigger`** attribute. Behaviors resulting from these particular values are illustrated in the series of figures that follow the table.

**Table 3-4.14 Possible Values within Transport Class Bits**

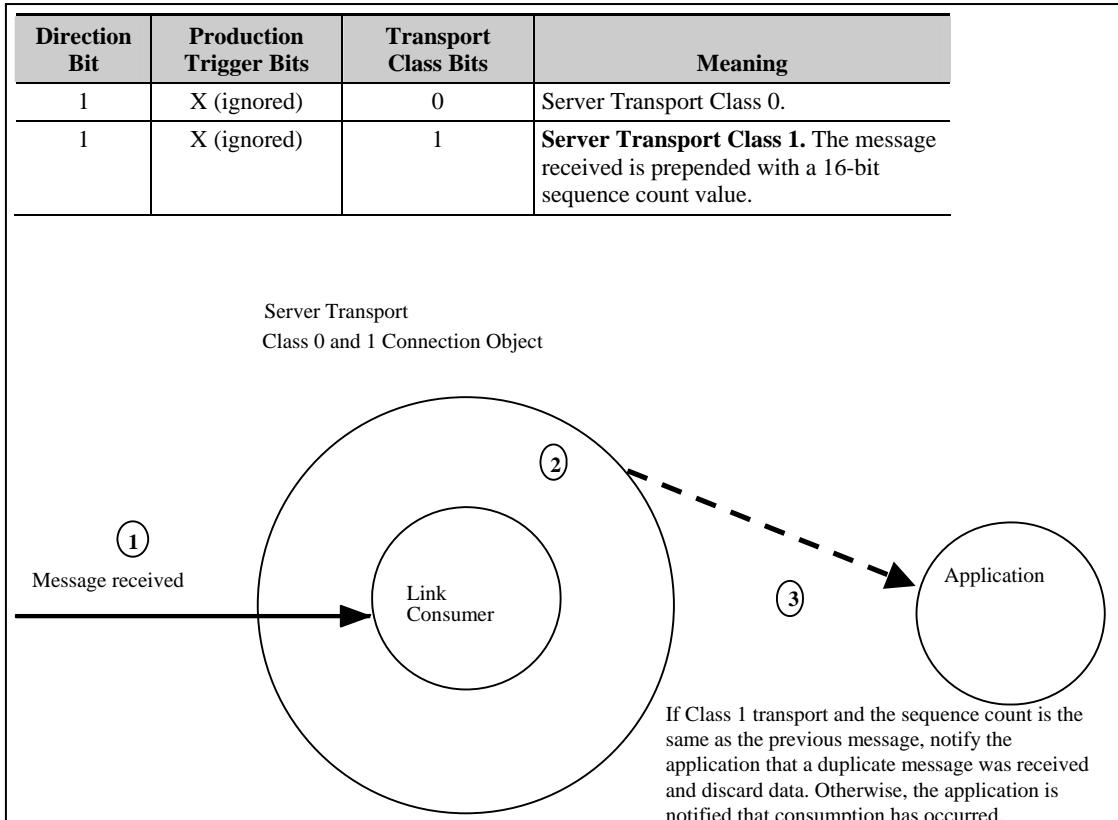
<b>Value</b>	<b>Meaning</b>	
0	Transport Class 0	Based on the value within the <code>Dir</code> bit, this connection end-point will be a producing only OR consuming only end-point. Upon application of this Connection instance, the module instantiates either a Link Producer ( <code>Dir</code> bit = Client, producing only) or a Link Consumer ( <code>Dir</code> bit = Server, consuming only) to be associated with this Connection.
1	Transport Class 1	
2	Transport Class 2	Indicates that the module will both produce AND consume across this connection. The Client end-point generates the first data production that is consumed by the Server, which causes the Server to return a production that is consumed by the Client.
3	Transport Class 3	
4	Transport Class 4	Non-blocking
5	Transport Class 5	Non-blocking, fragmenting
6	Transport Class 6	Multicast, fragmenting
7 - F	Reserved	

A 16-bit sequence count value is prepended to all Class 1, 2, and 3 transports. This value is used to detect delivery of duplicate data packets. Sequence count values are initialized on the first message production and incremented on each subsequent new data production. A resend of old data shall not cause the sequence count to change, and a consumer shall ignore data when it is received with a duplicate sequence count. Consuming applications can use this mechanism to distinguish between new samples and old samples that were sent to maintain the connection.

The following tables and figures illustrate the valid combinations of Production Trigger and Transport Class and provides a description of the Client and Server behaviors. See section 3-4.5.1 for a description of the Transmission Trigger Timer, which is shown in the illustrations.

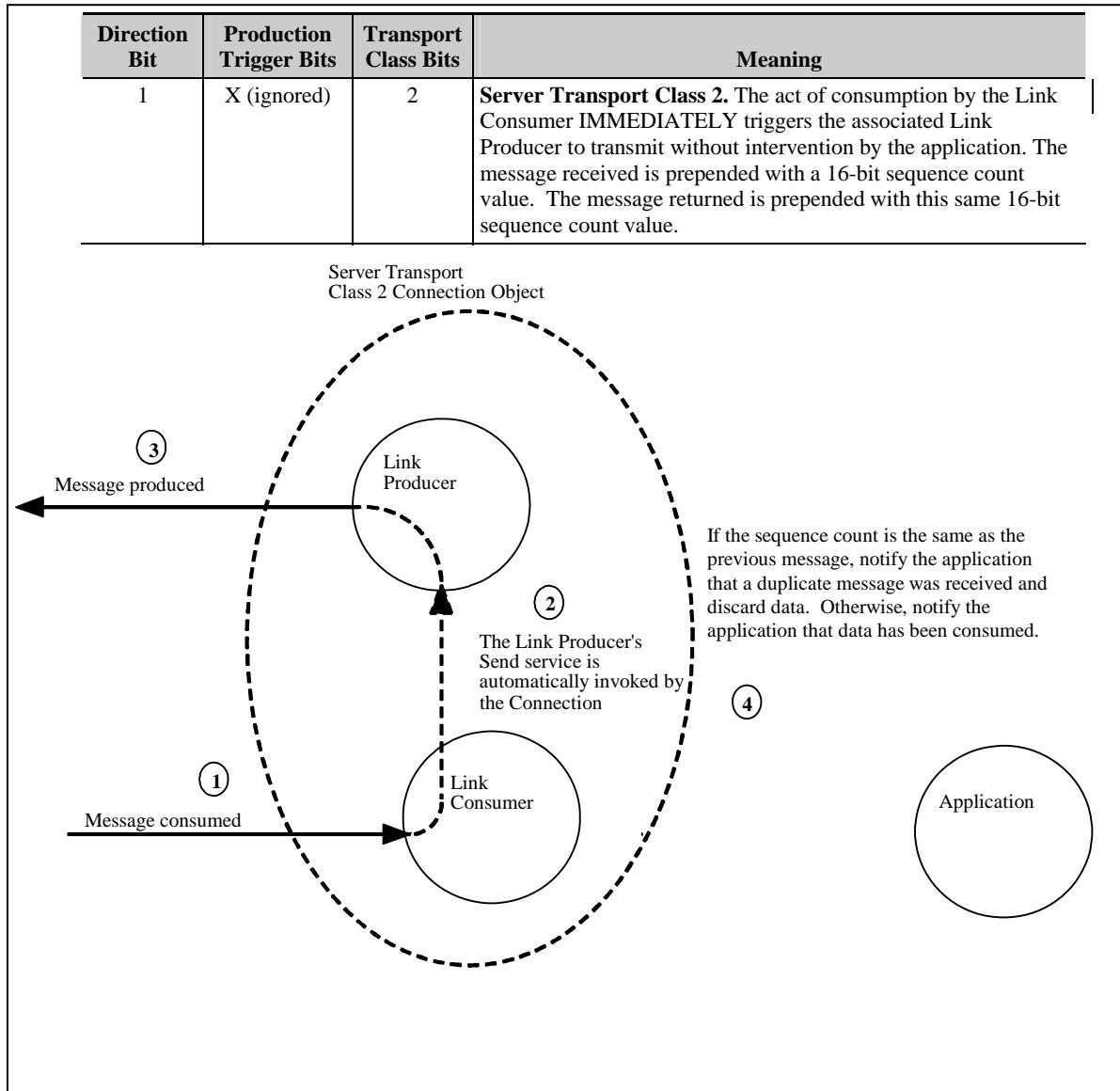
### 3-4.4.3.1 Server Transport Class 0 and 1 Behavior

**Figure 3-4.3 Server Transport Class 0 and 1 Behavior**



### 3-4.4.3.2 Server Transport Class 2 Behavior

Figure 3-4.4 Server Transport Class 2 Connection Object



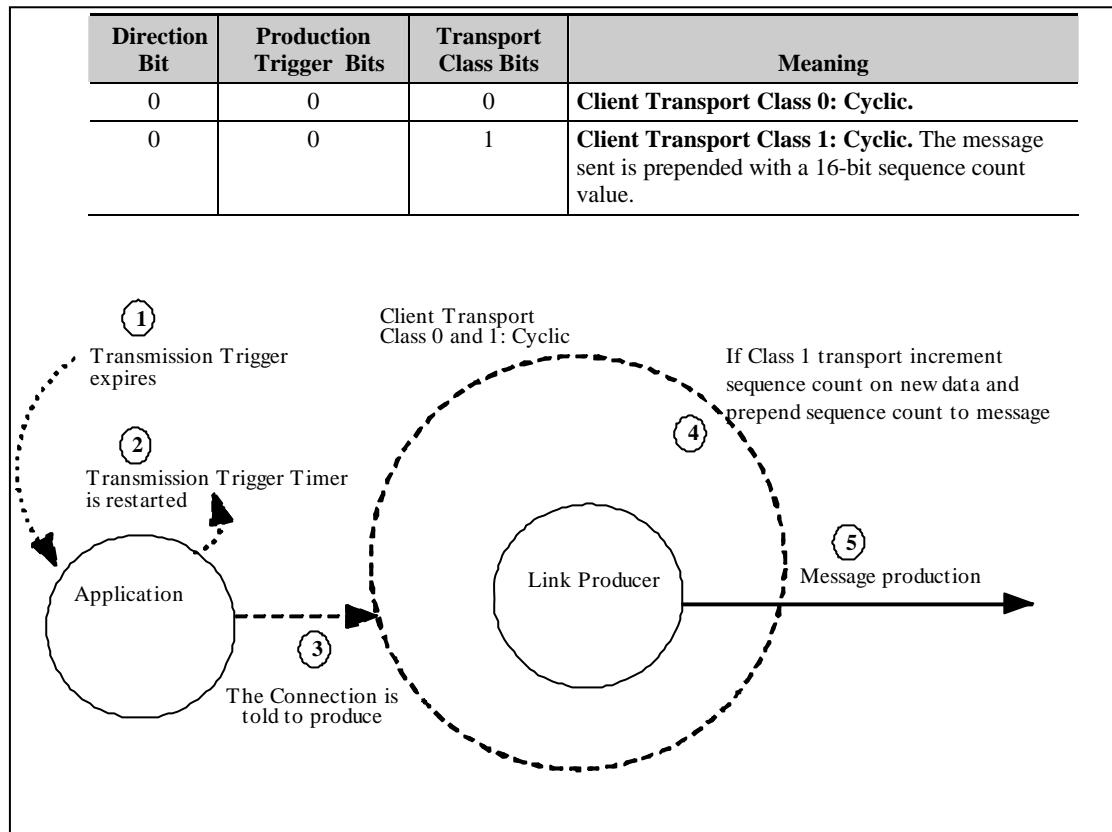
### 3-4.4.3.3 Server Transport Class 3 Behavior

Figure 3-4.5 Server Transport Class 3 Behavior

Direction Bit	Production Trigger Bits	Transport Class Bits	Meaning
1	X (ignored)	3	<p><b>Server Transport Class 3.</b> When the Link Consumer receives a message, it delivers it to the Application Object specified within the <b>consumed_connection_path</b> attribute. The Application Object then validates this <i>receive data</i> event (illustrated as #2 in the figure below). If the Application Object determines that the <i>receive data</i> event is valid, then it is REQUIRED to trigger a production as illustrated below. If the Application detects an error, then it may or may not trigger a production based on its own internal logic. The message received is prepended with a 16-bit sequence count value. The message returned is prepended with this same 16-bit sequence count value.</p> <pre> graph LR     LP((Link Producer)) -- "Message produced" --&gt; STCOC((Server Transport Class 3 Connection Object))     STCOC -- "The Connection is told to produce" --&gt; APP((Application))     APP -- "Message consumed" --&gt; LC((Link Consumer))     LC -- "If the sequence count is the same as the previous message, notify the application that a duplicate message was received and discard data. Otherwise, notify the application that data has been consumed." --&gt; APP     </pre> <p>The diagram illustrates the flow of data between four components: Link Producer, Link Consumer, Application, and Server Transport Class 3 Connection Object.      1. A solid arrow labeled '(1)' points from the Link Consumer to the Link Producer, labeled 'Message consumed'.     2. A dashed arrow labeled '(2)' points from the Link Consumer to the Application, containing the text: 'If the sequence count is the same as the previous message, notify the application that a duplicate message was received and discard data. Otherwise, notify the application that data has been consumed.'.     3. A dashed arrow labeled '(3)' points from the Application back to the Link Consumer, labeled 'The Connection is told to produce'.     4. A solid arrow labeled '(4)' points from the Link Producer to the Link Consumer, labeled 'Message produced'.</p>

### 3-4.4.3.4 Client Transport Class 0 and 1 Behavior: Cyclic

Figure 3-4.6 Client Transport Class 0 and 1 Behavior: Cyclic

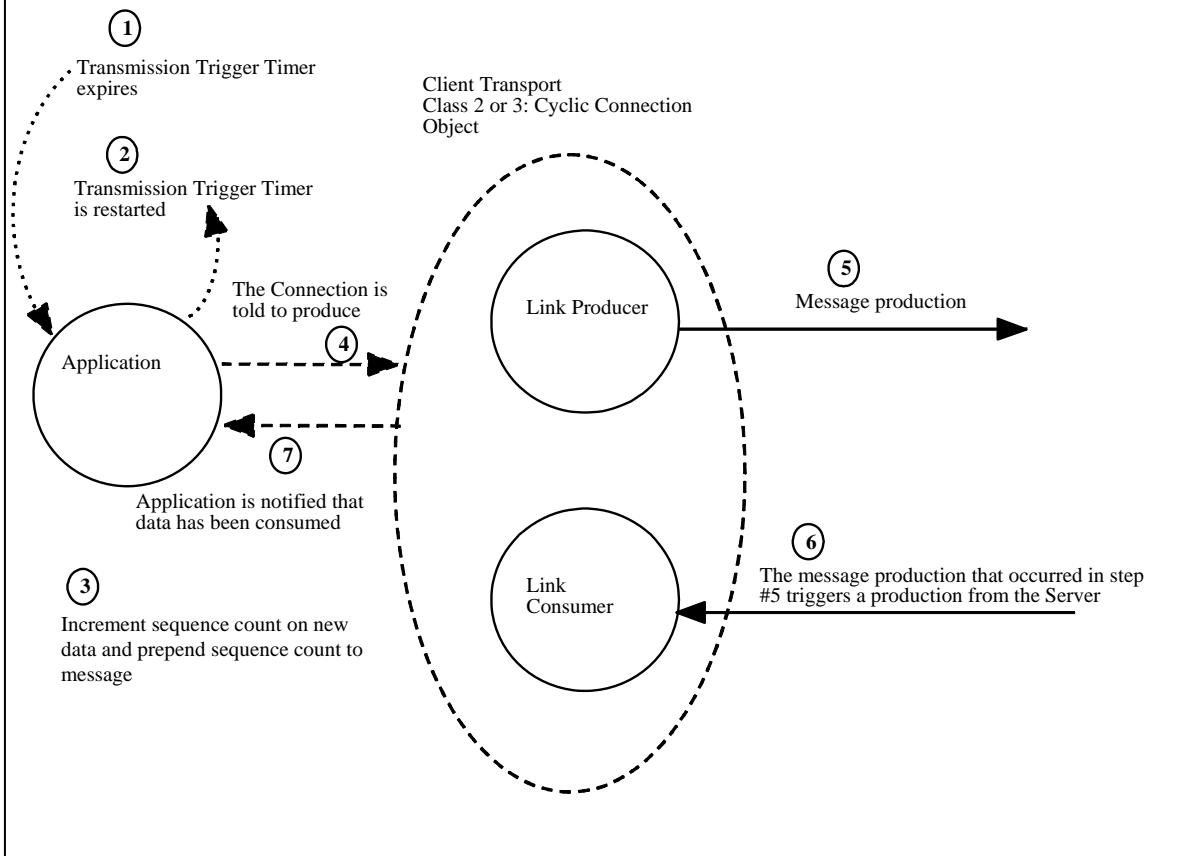


### 3-4.4.3.5 Client Transport Class 2 and 3 Behavior: Cyclic

Figure 3-4.7 Client Transport Classes 2 &amp; 3 Behavior: Cyclic

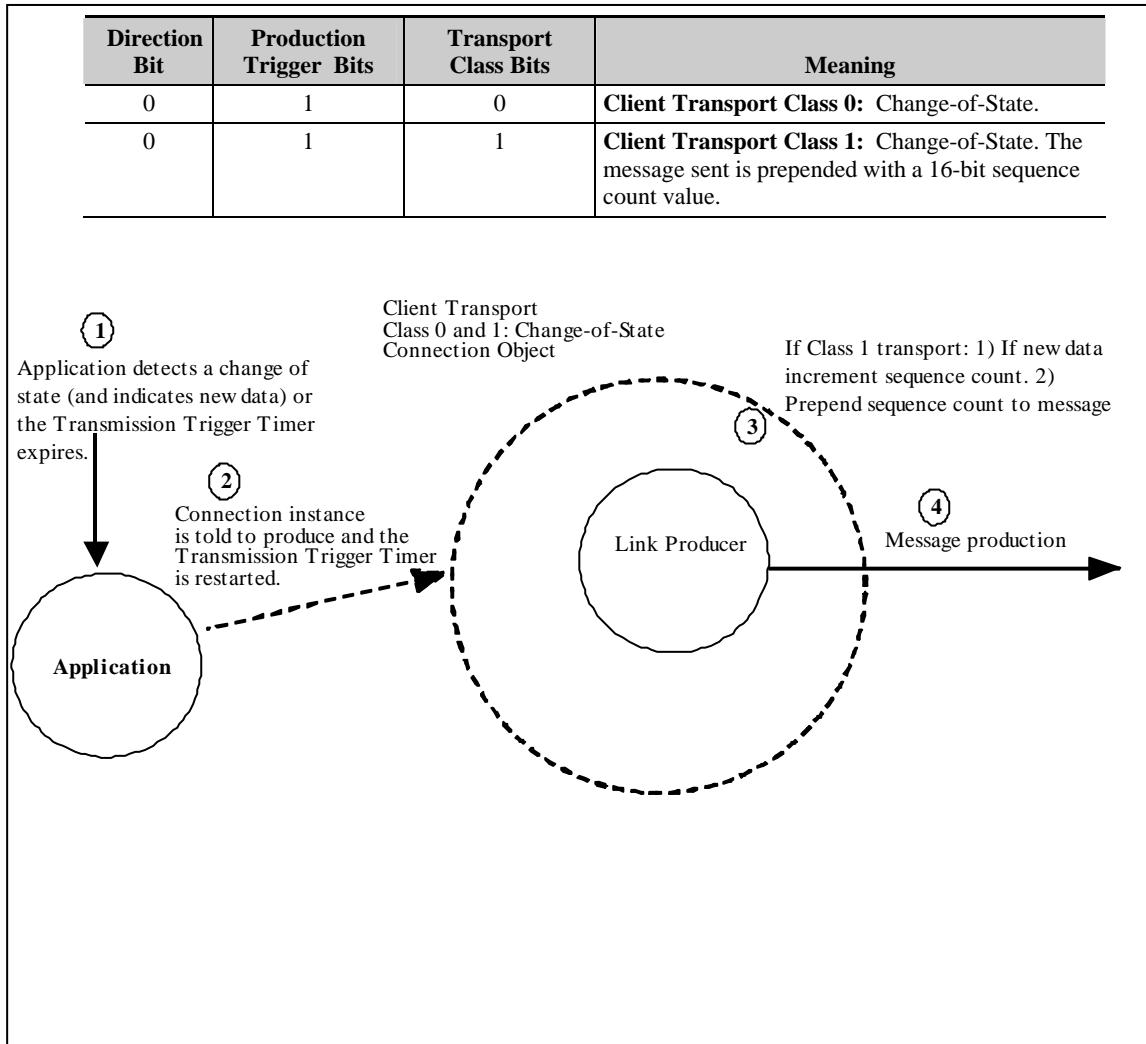
Direction Bit	Production Trigger Bits	Transport Class Bits	Meaning
0	0	2	Client Transport Class 2: Cyclic
0	0	3	Client Transport Class 3: Cyclic

For both transport class 2 and 3, a 16-bit sequence count value is prepended to the message before transmission. The message consumed due to this production is also prepended with a 16 bit sequence count with a value equal to that of the sequence count sent.



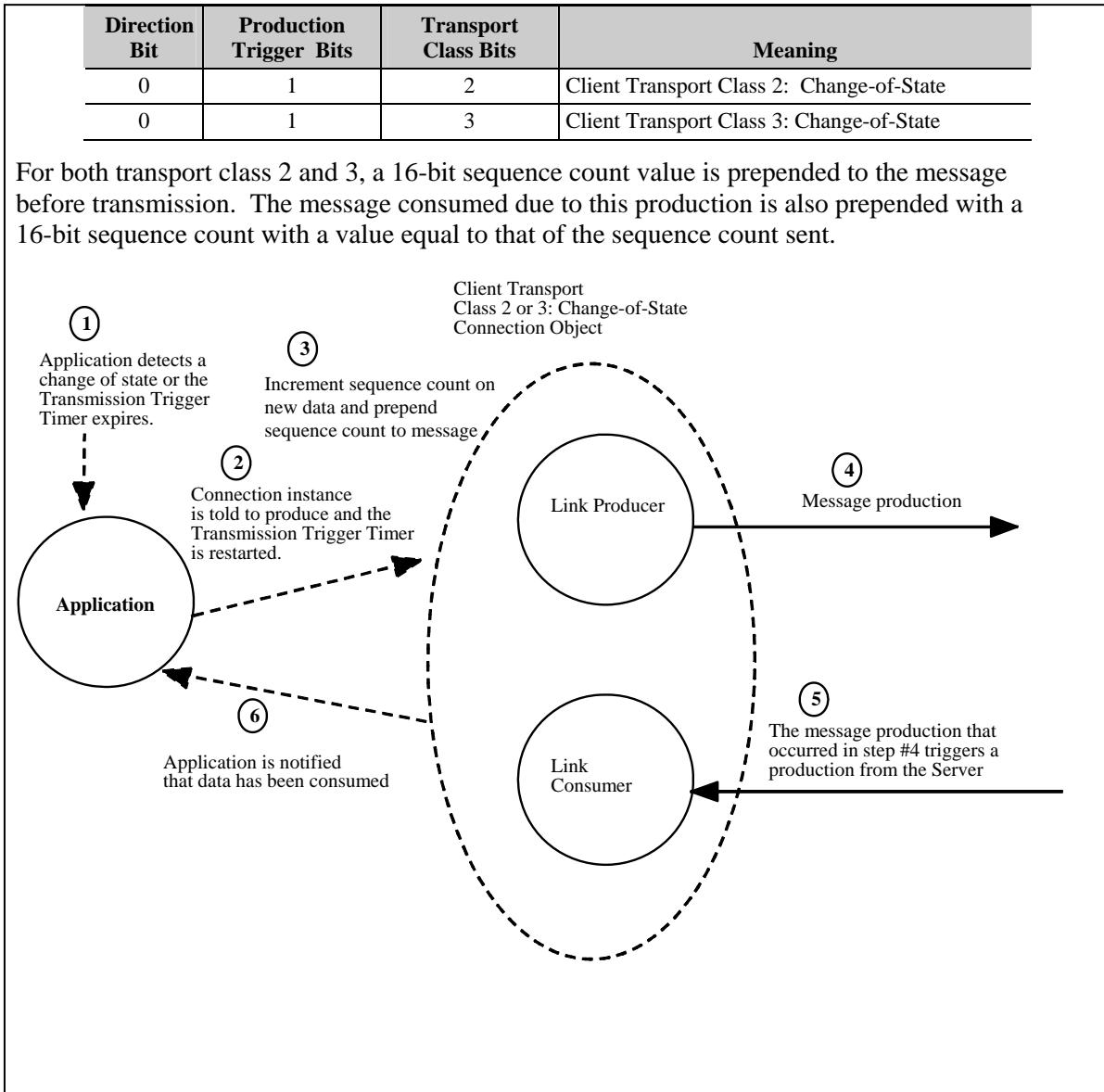
### 3-4.4.3.6 Client Transport Class 0 and 1 Behavior: Change-Of-State

Figure 3-4.8 Client Transport Class 0 and 1 Behavior: Change-Of-State



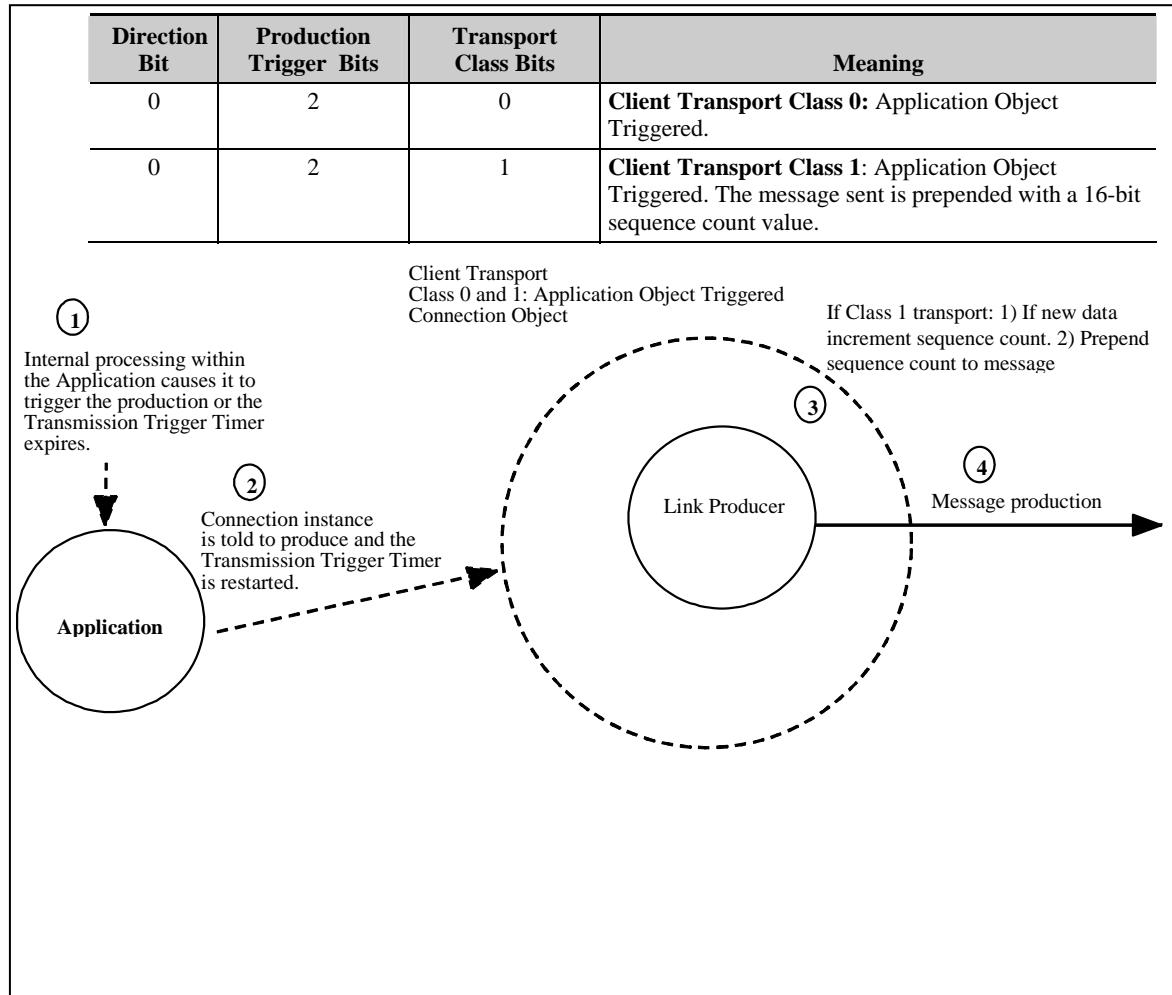
### 3-4.4.3.7 Client Transport Class 2 and 3 Behavior: Change-Of-State

Figure 3-4.9 Client Transport Classes 2 &amp; 3 Behavior: Change-Of-State



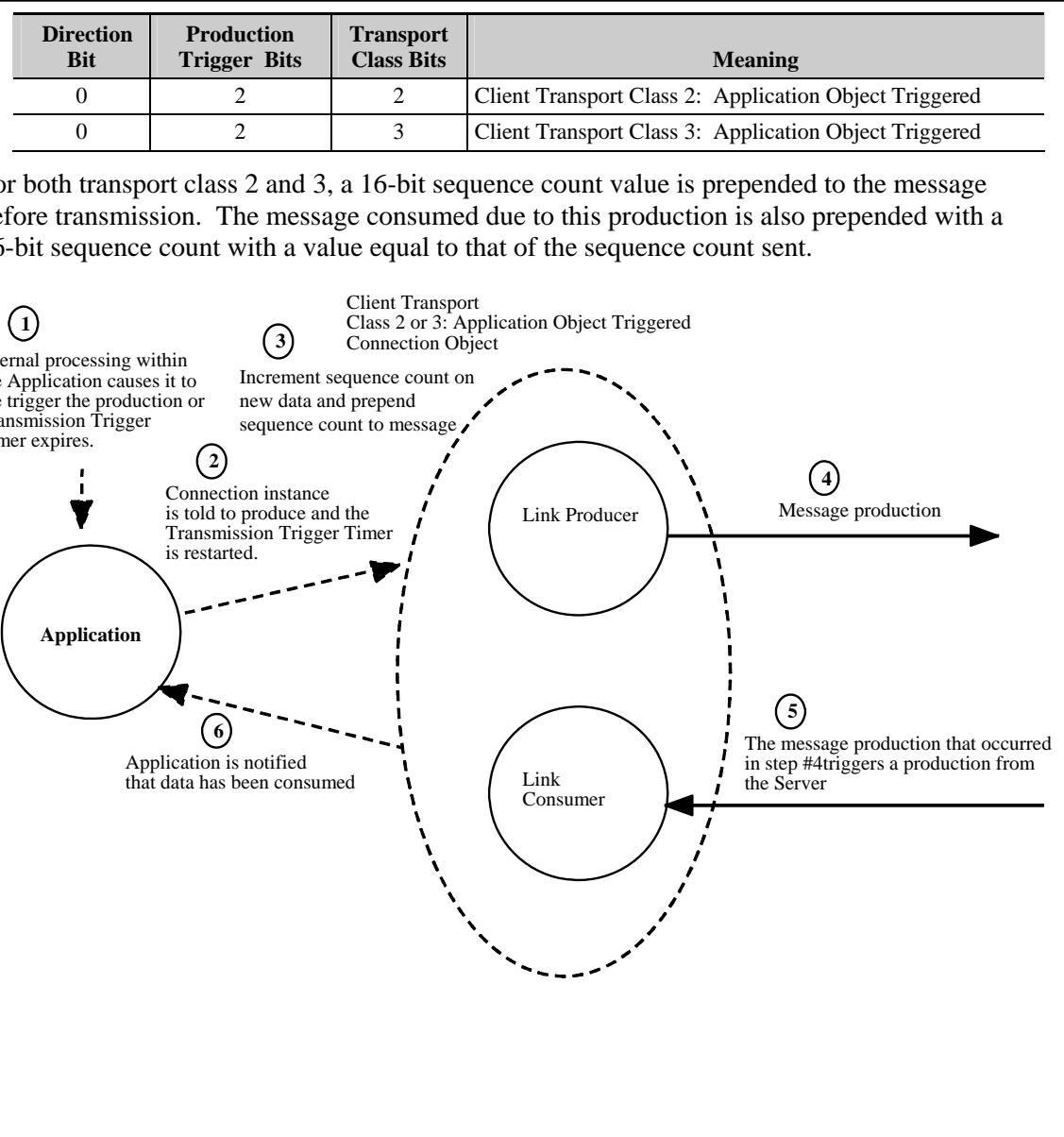
### 3-4.4.3.8 Client Transport Class 0 and 1 Behavior: Application Object Triggered

Figure 3-4.10 Client Transport Class 0 and 1 Behavior: Application Object Triggered



### 3-4.4.3.9 Client Transport Class 2 and 3 Behavior: Application Object Triggered

Figure 3-4.11 Client Transport Classes 2 &amp; 3 Behavior: Application Object Triggered



To summarize, refer to the following table for the valid values within the **transportClass\_trigger** attribute of a Connection Instance:

**Table 3-4.15 Transport Class\_Trigger Attribute**

TransportClass_trigger bits	Meaning
1 xxx 0000	Direction = Server, Production Trigger = IGNORED, Transport Class = 0.
1 xxx 0001	Direction = Server, Production Trigger = IGNORED, Transport Class = 1.
1 xxx 0010	Direction = Server, Production Trigger = IGNORED, Transport Class = 2.
1 xxx 0011	Direction = Server, Production Trigger = IGNORED, Transport Class = 3. This is the value assigned to this attribute within the Server end-point of an Explicit Messaging Connection.
0 000 0000	Direction = Client, Production Trigger = Cyclic, Transport Class = 0.
0 000 0001	Direction = Client, Production Trigger = Cyclic, Transport Class = 1.
0 000 0010	Direction = Client, Production Trigger = Cyclic, Transport Class = 2.
0 000 0011	Direction = Client, Production Trigger = Cyclic, Transport Class = 3.
0 001 0000	Direction = Client, Production Trigger = Change-Of-State, Transport Class = 0.
0 001 0001	Direction = Client, Production Trigger = Change-Of-State, Transport Class = 1.
0 001 0010	Direction = Client, Production Trigger = Change-Of-State, Transport Class = 2.
0 001 0011	Direction = Client, Production Trigger = Change-Of-State, Transport Class = 3.
0 010 0000	Direction = Client, Production Trigger = Application Object, Transport Class = 0.
0 010 0001	Direction = Client, Production Trigger = Application Object, Transport Class = 1
0 010 0010	Direction = Client, Production Trigger = Application Object, Transport Class = 2.
0 010 0011	Direction = Client, Production Trigger = Application Object, Transport Class = 3. This is the value assigned to this attribute within the Client end-point of an Explicit Messaging Connection.
1 111 1111	Default value assigned to this attribute within an I/O Connection.

#### **3-4.4.4 DeviceNet\_produced\_connection\_id, Attribute 4 – UINT data type**

This attribute is Required for a DeviceNet subnet. Other subnet types shall not use this attribute. See Volume 3, DeviceNet Adaptation of CIP, Chapter 3 for an explanation of this attribute.

#### **3-4.4.5 DeviceNet\_consumed\_connection\_id, Attribute 5 – UINT data type**

This attribute is Required for a DeviceNet subnet. Other subnet types shall not use this attribute. See Volume 3, DeviceNet Adaptation of CIP, Chapter 3 for an explanation of this attribute.

#### **3-4.4.6 DeviceNet\_initial\_comm\_characteristics, Attribute 6 – USINT data type**

This attribute is Required for a DeviceNet subnet. Other subnet types shall not use this attribute. See Volume 3, DeviceNet Adaptation of CIP, Chapter 3 for an explanation of this attribute.

### **3-4.4.7    Produced\_connection\_size, Attribute 7 – UINT data type**

The meaning of this attribute is different for Explicit Messaging Connections than it is for I/O Connections. If the subnet defines a fragmentation protocol and the device supports fragmentation, this size may be larger than the largest frame size. See the network specific adaptation specifications for more details.

#### *For Explicit Messaging Connections:*

This attribute signifies the maximum number of Message Router Request/Response Data octets (See CIP Common, Chapter 2) that a device is able to transmit across this Connection. Devices that place a known limit on the maximum amount of Message Router Request/Response Data that can be transmitted in a single message, or single fragmented series initialize this attribute accordingly. Devices that cannot or do not predefine an up-front transmit limit place the value 0xffff into this attribute (there may still be a limit, however, it is not known in advance).

**Important:** Due to the nature of Explicit Messaging, the length of Explicit Messages will fluctuate over the lifetime of a connection. Explicit Messaging Connections perform fragmentation based on the *length* of the current message to transmit where the subnet type may define a fragmentation protocol.

#### *For I/O Connections:*

If the transportClass\_trigger indicates that this Connection instance is to produce, then this attribute defines the maximum amount of I/O data that may be produced as a single unit across this connection. The amount of I/O to be transmitted at any given point in time can be less than or equal to the connection\_size attribute.

This attribute defaults to zero (0) within an I/O Connection. If the subnet type supports fragmentation and this attribute is set to a value greater than the largest payload in an I/O Connection, then the Connection will break up the data into multiple fragments. See the network specific adaptation specifications for more details.

**Important:** Fragmentation within I/O Connections is performed based on the value within this attribute, regardless of the current amount of data to transmit.

**Important:** I/O Messages that contain no Application I/O Data and were configured to contain data (via produced\_connection\_size being greater than zero (0)) are defined to indicate a *No Data* event for the receiving Application Object(s). The behavior of an Application Object upon detection of the *No Data* event is Application Object specific.

### **3-4.4.8    Consumed\_connection\_size, Attribute 8 – UINT data type**

The meaning of this attribute is different for Explicit Messaging Connections than it is for I/O Connections. If the subnet defines a fragmentation protocol and the device supports fragmentation, this size may be larger than the largest frame size. See the network specific adaptation specifications for more details.

#### *For Explicit Messaging Connections:*

This attribute signifies the maximum number of Message Router Request/Response Data octets (See CIP Common, Chapter 2) that a device is able to receive across this Connection.

Devices that place a known limit on the maximum amount of Message Router Request/Response Data that can be received in a single message, or single fragmented series initialize this attribute accordingly. Devices that cannot or do not predefine an up-front receive limit place the value 0xffff into this attribute (there may still be a limit, however, it is not known in advance). Because of the nature of Explicit Messaging, the length of Explicit Messages will fluctuate over the lifetime of a connection.

**For I/O Connections:**

If the transportClass\_trigger attribute indicates that this Connection is to consume, then this attribute defines the maximum amount of data that may be received as a single unit across this connection. The actual amount of I/O data received at any given time can be less than or equal to the connection\_size attribute.

This attribute defaults to zero (0) within an I/O Connection. If the subnet type supports fragmentation and this attribute is set to a value greater than the largest payload in an I/O Connection, then the Connection will process the fragmentation protocol. See the network specific adaptation specifications for more details.

The length of an I/O Message must be less than or equal to this attribute for an I/O Connection Object to receive it as a valid message. If an I/O Connection Object receives a message whose length is greater than this attribute, then it immediately discards the message and discontinues any subsequent processing. Note that with respect to the Server end-point of a Transport Class 2 or 3 Connection, this too much data error condition results in no response being transmitted and the watchdog timer is not kicked.

#### **3-4.4.9    Expected\_packet\_rate, Attribute 9 – UINT data type**

This attribute is used to generate the values loaded into the *Transmission Trigger Timer* and the *Inactivity/Watchdog Timer*. See section 3-4.5, Connection Timing for a description of the Transmission Trigger and Inactivity/Watchdog timers.

The resolution of this attribute is in milliseconds. A request to configure this attribute may result in the specification of a time value that a product cannot meet. In addition to performing product specific range checking when a request to modify this attribute is received, the following steps are performed:

- If the specified value is not equal to an increment of the available clock resolution, then the value is rounded up to the next serviceable value. For example: a Set\_Attribute\_Single request is received specifying the value 5 for the **expected\_packet\_rate** attribute and the product provides a 10 millisecond resolution on timers. In this case the product would load the value 10 into the **expected\_packet\_rate** attribute.
- The value that is actually loaded into the **expected\_packet\_rate** attribute is reported in the Service Data Field of a Set\_Attribute\_Single response message associated with a request to modify this attribute.
- If the requested value is equal to an increment of the clock resolution, then the requested value is loaded into the **expected\_packet\_rate** and reported in the response. For example: if the value 100 is requested and the clock resolution is 10 milliseconds, then a value of 100 is loaded.

When a Connection Object is in the Established state, any modifications to the **expected\_packet\_rate** attribute have immediate effect on the Inactivity/Watchdog Timer. The following steps are performed by a Connection Object in the Established state when a request is received to modify the **expected\_packet\_rate** attribute:

- the current Inactivity/Watchdog Timer is canceled
- a new Inactivity/Watchdog Timer is activated based on the new value in the **expected\_packet\_rate** attribute.

This attribute defaults to 2500 (2500 milliseconds) within Explicit Messaging Connections, and to zero (0) within an I/O Connection.

#### **3-4.4.10 CIP\_produced\_connection\_id, Attribute 10 – UDINT data type**

Contains the Connection ID, which identifies messages to be sent across this connection (if any). This attribute shall not be implemented when the subnet type is DeviceNet.

#### **3-4.4.11 CIP\_consumed\_connection\_id, Attribute 11 – UDINT data type**

Contains the Connection ID, which identifies messages to be received across this connection (if any). This attribute shall not be implemented when the subnet type is DeviceNet.

#### **3-4.4.12 Watchdog\_timeout\_action, Attribute 12 – USINT data type**

This attribute defines the action the Connection Object should perform when the Inactivity/Watchdog Timer expires. The table below defines the specifics of this attribute.

**Table 3-4.16 Values for the watchdog\_timeout\_action**

Value	Meaning
0	<i>Transition to Timed Out.</i> The Connection transitions to the Timed Out state and remains in this state until it is Reset or Deleted. The command to Reset or Delete could come from an internal source (e.g., an Application Object) or could come from the network (e.g., a configuration tool). This is the default value for this attribute with respect to I/O Connections. This value is invalid for Explicit Messaging Connections.
1	<i>Auto Delete.</i> The Connection Class automatically deletes the Connection if it experiences an Inactivity/Watchdog timeout. This is the default value for this attribute with respect to Explicit Messaging Connections.
2	<i>Auto Reset.</i> The Connection remains in the Established state and immediately restarts the Inactivity/Watchdog timer. This value is invalid for Explicit Messaging Connections.
3	<i>Deferred Delete.</i> The Connection transitions to the Deferred state if any child connection instances are in the Established state. If no child connection instances are in the Established state the connection is deleted. This value is only used on DeviceNet and is invalid for I/O Messaging Connections.
4 - FF	Reserved by CIP

#### **3-4.4.13 Produced\_connection\_path\_length, Attribute 13 – UINT data type**

Specifies the number of bytes of information within the **produced\_connection\_path** attribute. This is automatically initialized when the **produced\_connection\_path** attribute is configured. This attribute defaults to the value zero (0).

**3-4.4.14 Produced\_connection\_path, Attribute 14 – EPATH data type**

The **produced\_connection\_path** attribute is made up of a byte stream which defines the Application Object(s) whose *data* is to be produced by this Connection Object. **The format of this byte stream is specified in Appendix C, Abstract Syntax Encoding for Segment Types.** This attribute defaults to being empty upon instantiation of the Connection. It remains empty within Explicit Messaging Connections and within Connection Objects that do not produce.

**3-4.4.15 Consumed\_connection\_path\_length, Attribute 15 – UINT data type**

Specifies the number of bytes of information within the **consumed\_connection\_path** attribute. This is automatically initialized when the **consumed\_connection\_path** attribute is configured. This attribute defaults to the value zero (0).

**3-4.4.16 Consumed\_connection\_path, Attribute 16 – EPATH data type**

The **consumed\_connection\_path** attribute is made up of a byte stream which defines the Application Object(s) that are to receive the *data* consumed by this Connection Object. **The format of this byte stream is specified in Appendix C, Abstract Syntax Encoding for Segment Types.** This attribute defaults to being empty upon instantiation of the Connection. It remains empty within Explicit Messaging Connections and within Connection Objects that do not consume.

**3-4.4.17 Production\_inhibit\_time, Attribute 17 – UINT data type**

This attribute is used to configure the minimum delay time between new data production. This is required for all I/O Client connections, except those with a production trigger of Cyclic. The Set\_Attribute\_Single service must be supported when this attribute is implemented. A value of zero (the default value for this attribute) indicates no inhibit time.

The resolution of this attribute is in milliseconds. A request to configure this attribute may result in the specification of time value that a product cannot meet. In addition to performing product specific range checking when a request to modify this attribute is received, the following steps are performed:

- If the specified value is not equal to an increment of the available clock resolution, then the value is rounded **up** to the next serviceable value. For example: a Set\_Attribute\_Single request is received specifying the value 5 for the **production\_inhibit\_time** attribute and the product provides a 10 millisecond resolution on times. In this case the product would load the value 10 into the **production\_inhibit\_time** attribute.
- The value that is actually loaded into the **production\_inhibit\_time** attribute is reported in the Service Data Field of a Set\_Attribute\_Single response message associated with a request to modify this attribute.
- If the requested value is equal to an increment of the clock resolution, then the requested value is loaded into the **production\_inhibit\_time** and reported in the response. For example: the value 100 is requested and the clock resolution is 10 milliseconds.

The **production\_inhibit\_time** value is loaded in the Production Inhibit Timer each time new data production occurs.

When a Connection Object is in the **Established** state, any modifications to the **production\_inhibit\_time** attribute have no effect on a currently running Production Inhibit Timer. The new production\_inhibit\_time value is loaded into the Production Inhibit Timer on the following new data production.

When the apply\_attributes service is received, the **production\_inhibit\_time** must be verified against the **expected\_packet\_rate** attribute. If the **expected\_packet\_rate** value is greater than zero, but less than the **production\_inhibit\_time** value, then an error shall be returned. In this case, where two attribute values conflict use the **production\_inhibit\_time** attribute ID as the additional error code returned in the error response.

#### **3-4.4.18 Connection\_timeout\_multiplier, Attribute 18 –USINT data type**

The Connection\_timeout\_multiplier specifies the multiplier applied to the expected packet rate to obtain the value used by the Inactivity/Watchdog Timer. See the Connection Timeout Multiplier parameter description within the Connection Manager Object Specific Service Parameters for the enumerated values of this attribute. The default value for this attribute is zero (specifying a multiplier of 4).

#### **3-4.4.19 Connection\_binding\_list, Attribute 19 – Struct(UINT(size), Array of UINT[size]) data type**

The Connection\_binding\_list attribute identifies connection instances that are bound to this connection. The attribute structure provides a 16 bit count of bound connections, followed by a list of the bound instances. This attribute shall be supported if the Connection Bind service is supported.

### **3-4.5 Connection Timing**

Three types of timers are involved in a connection:

- Transmission Trigger Timer
- Inactivity/Watchdog Timer
- Production Inhibit Timer

The first two timers are initialized based on the value in the **expected\_packet\_rate** attribute.

**Important:** For Explicit Messaging, the Application is responsible for providing *response timeout* facilities. The amount of time a Client waits for a Server to respond to a request depends on the application and, possibly, on the service. Due to Media Access mechanisms used for accessing the subnet, an implementation should wait until an Explicit Request Message is transmitted before activating any response related timers.

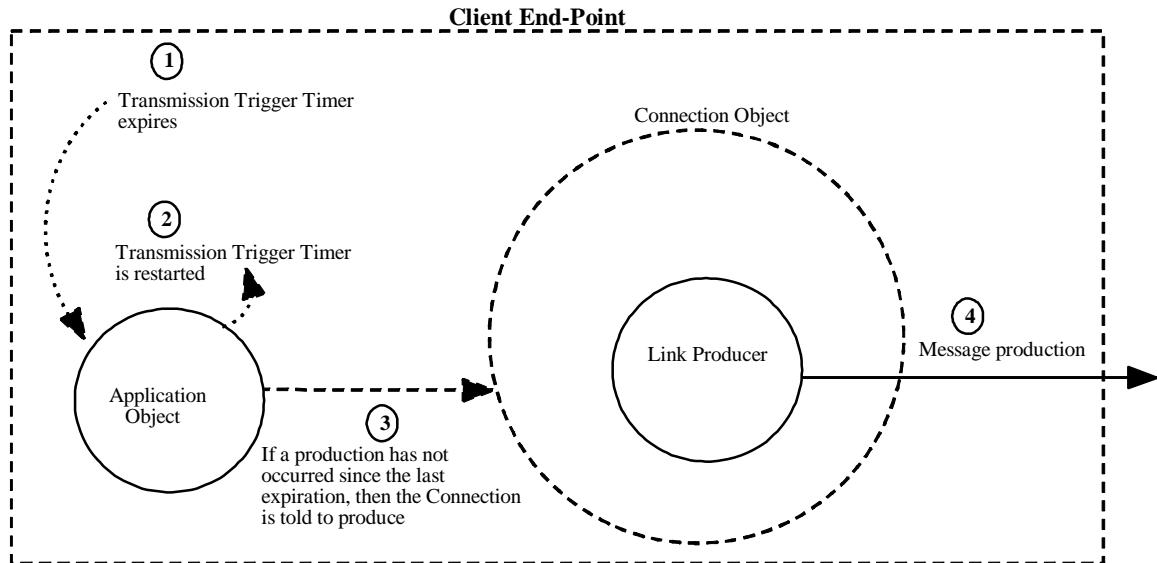
#### **3-4.5.1 Transmission Trigger Timer**

This timer is required to be managed by the application within the Client end-point of a Connection. Expiration of this timer is an indication that the associated Connection Object may need to be told to transmit a message. If a production has not occurred since the timer was activated, then the Connection Object should be told to produce to avoid an Inactivity/Watchdog timeout at the Server end-point(s).

The tasks listed below are performed by a Connection immediately upon producing a message

- The current value of the Transmission Trigger Timer is restored to its initial value and the timer is stopped.
- A new Transmission Trigger Timer is activated.

**Figure 3-4.12 Transmission Trigger Timer**



As illustrated in Figure 3-4.12, when the Transmission Trigger Timer expires, it is immediately re-started. This timer is activated when the Connection transitions to the **Established** state.

**Important:** The Transmit Trigger Timer is initialized with the value in the `expected_packet_rate` attribute. If the `expected_packet_rate` attribute contains the value zero (0), then the Transmission Trigger Timer is not activated and/or used by the Client end-point.

**Important:** Server end-points do not activate this timer.

### 3-4.5.2 Inactivity/Watchdog Timer

This timer is **required** to be managed by any **consuming** Connection Object. Consuming Connection Objects include:

- Client end-point Connection Objects whose `transportClass_trigger` attribute indicates either Transport Class 2 or 3
- All Server end-point Connection Objects

This timer is activated when the Connection transitions to the **Established** state. The tasks listed below are performed by a Connection immediately upon detecting that a **valid** message has been consumed:

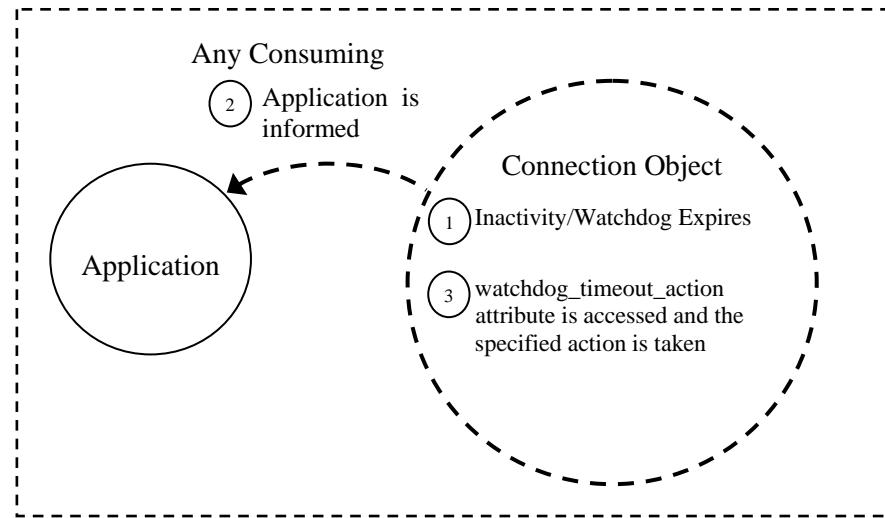
- The current value for the Inactivity/Watchdog Timer is restored to its initial value and the timer is stopped.
- A new Inactivity/Watchdog Timer is activated.

The bullet items above indicate that the new Inactivity/Watchdog Timer is activated before the received message is processed.

Expiration of this timer is an indication that the Connection Object has timed out while waiting to consume. The Connection Object performs the following steps when the Inactivity/Watchdog timer expires:

- issues an indication of this event to the application
- performs the action indicated by the `watchdog_timeout_action` attribute

**Figure 3-4.13 Inactivity Watchdog Timer**

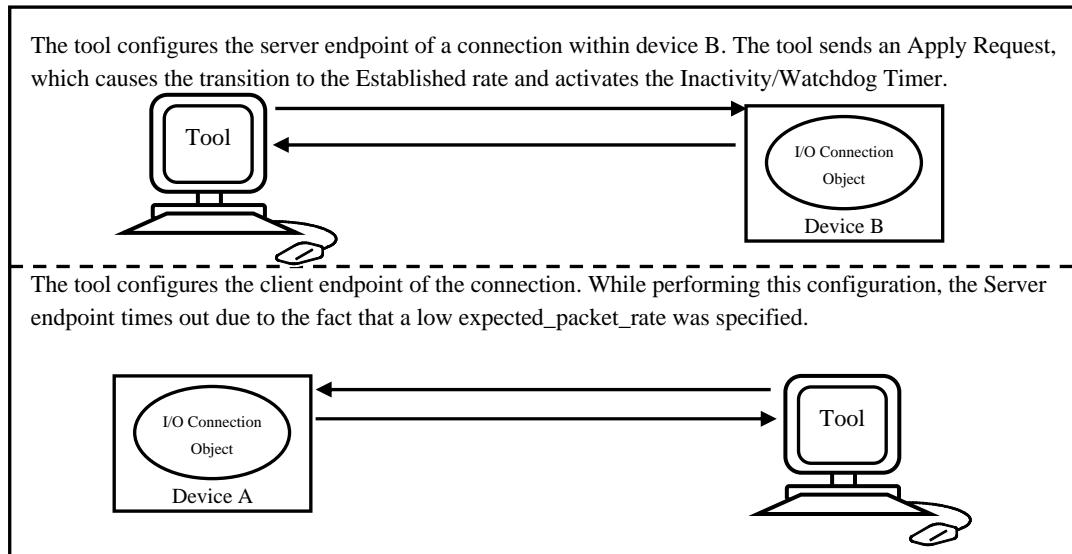


Two different values are used for the Inactivity/Watchdog Timer, based on whether or not the Connection Object has consumed a message:

1. The **initial** value loaded into the Inactivity/Watchdog Timer is either 10,000 milliseconds (10 seconds) or the **expected\_packet\_rate** multiplied by the **connection\_timeout\_multiplier**, depending on which value is numerically greater<sup>1</sup>. If the **expected\_packet\_rate** attribute multiplied by the **connection\_timeout\_multiplier** is greater than 10,000, then the **expected\_packet\_rate** multiplied by the **connection\_timeout\_multiplier** is used. Otherwise, 10,000 (10 seconds) is used. This is referred to as the **pre-consumption** timeout value. This value is used because a Connection may transition to the **Established** state before all end-points are fully configured. For example:

---

1 If the **expected\_packet\_rate** attribute contains the value zero (0), then the Inactivity/Watchdog Timer is not activated and/or used by the Connection Object. A Connection Object whose **expected\_packet\_rate** attribute is zero (0) will never experience an Inactivity/Watchdog timeout.



This rule takes into consideration that the application-to-application connection is not really established until the associated information exchange is performed for the first time.

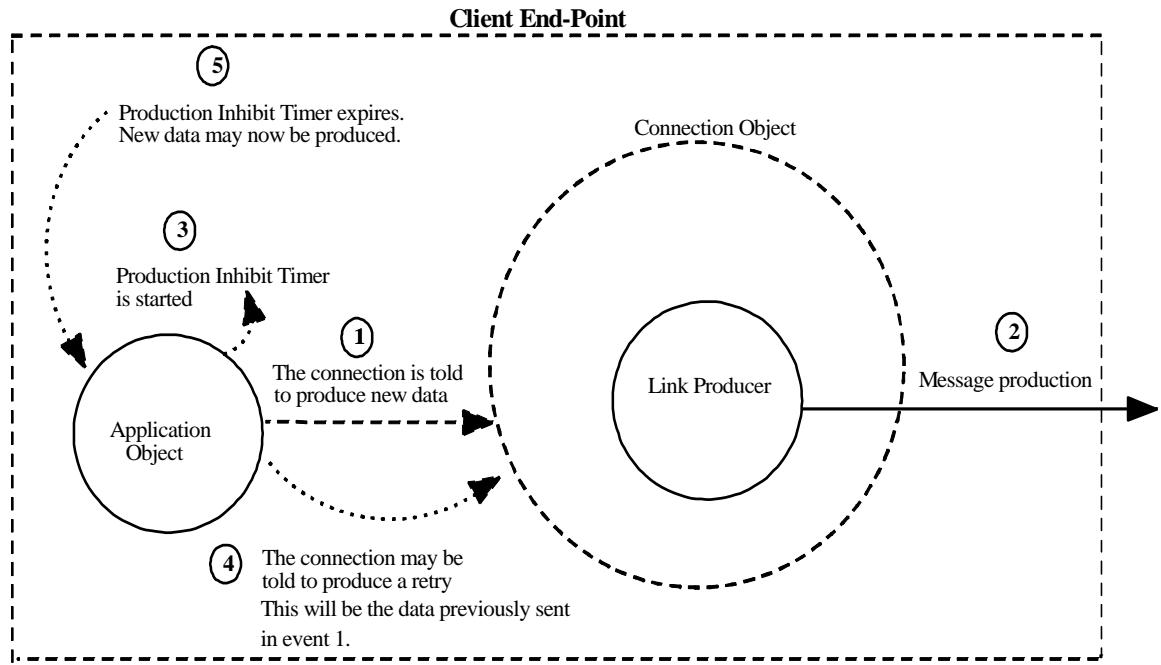
2. All subsequent activations of the Inactivity/Watchdog Timer use the **expected\_packet\_rate** multiplied by the **connection\_timeout\_multiplier** as the number of milliseconds to load into the Inactivity/Watchdog Timer 1.

### 3-4.5.3 Production Inhibit Timer

This timer is **required** to be managed by the Connection Object within the **Client end-point** of an I/O Connection when the **production\_inhibit\_time** attribute value is non-zero. This timer is started when data is produced by the Connection Object. The Connection Object may not produce new data if this timer is running. A retry may, however, be sent to the Link Producer. Expiration of this timer allows the Connection Object to send new data.

- 
- 1 If the **expected\_packet\_rate** attribute contains the value zero (0), then the Inactivity/Watchdog Timer is not activated and/or used by the Connection Object. A Connection Object whose **expected\_packet\_rate** attribute is zero (0) will never experience an Inactivity/Watchdog timeout.

Figure 3-4.14 Production Inhibit Timer



**Important:** The Production Inhibit Timer is initialized with the value in the **production\_inhibit\_time** attribute. If the **production\_inhibit\_time** attribute contains the value zero (0), then the Production Inhibit Timer is not activated and/or used by the Client end-point.

**Important:** Server end-points do not activate this timer.

### 3-4.6 Connection Object Instance Services

The Connection Object Instance supports the following CIP Common services:

Table 3-4.17 Connection Object Instance Services

Service Code	Need In Implementation	Service Name	Service Description
0Ehex	Required	Get_Attribute_Single	Used to read a Connection Object attribute.
10hex	Optional	Set_Attribute_Single	Used to modify a Connection Object attribute. The Connection Object returns information in the Service Data Field of a Set_Attribute_Single Response when the <b>expected_packet_rate</b> attribute is modified as indicated in Table 3-4.19
05hex	Optional	Reset	Used to reset the Inactivity/Watchdog Timer associated with a Connection Object. When a Connection in the <b>Timed Out</b> or Deferred Delete state receives a Reset request it also transitions back to the <b>Established</b> state.
09hex	Optional	Delete	Used to delete a Connection Object and to release all associated resources.

Service Code	Need In Implementation	Service Name	Service Description
0Dhex	Optional	Apply_Attributes	Used to deliver the Connection Object to the application, which performs the set of tasks necessary to create the specified connection. A Connection Instance returns the <b>produced_connection_id</b> and <b>consumed_connection_id</b> attributes within the Service Data field of an Apply Attributes Response Message as indicated in Table 3-4.18. The Apply_Attributes service effectively states that the initial configuration of the Connection Object is complete and it is now time to "activate" that configuration.

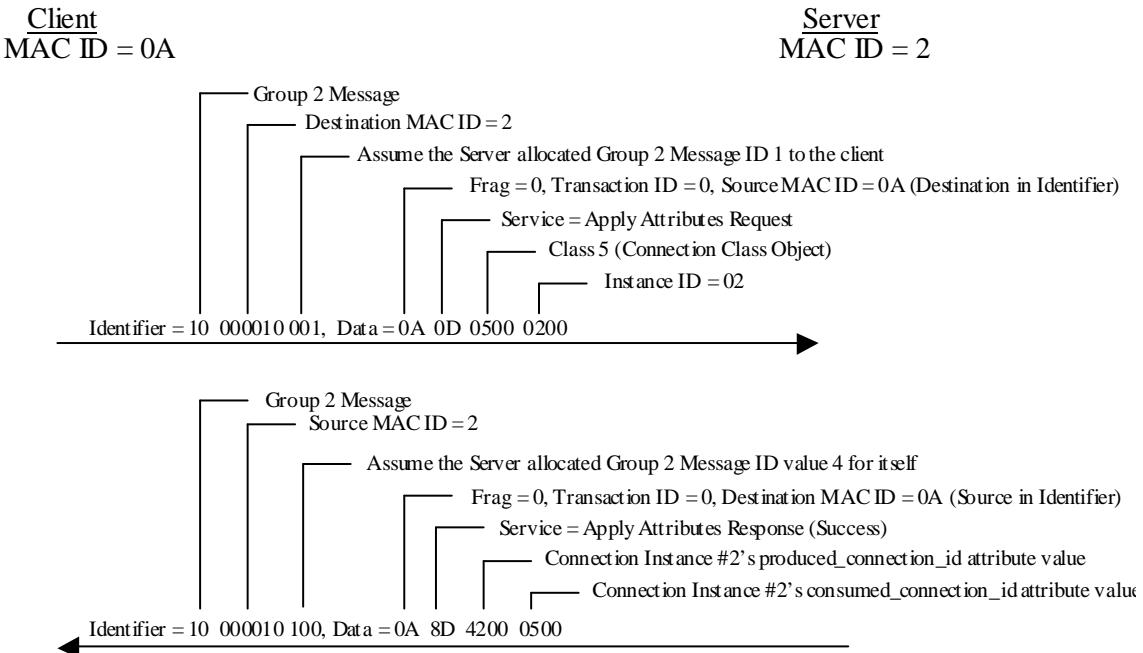
The following information is specified by a Connection Object Instance within the Service Data Field of a successful Apply\_Attributes response.

**Table 3-4.18 Service Data For Connection Object Apply Attributes Response**

Name	Data Type	Description of Parameter
Produced Connection ID	UINT	Contains the value within the Connection Instance's <b>produced_connection_id</b> attribute. If the subnet type is DeviceNet, the <b>DeviceNet_produced_connection_id</b> attribute is used.
Consumed Connection ID	UINT	Contains the value within the Connection Instance's <b>consumed_connection_id</b> attribute. If the subnet type is DeviceNet, the <b>DeviceNet_consumed_connection_id</b> attribute is used.

Figure 3-4.15 below illustrates the Apply\_Attributes service sent to a Connection Object Instance. For this example, assume a DeviceNet Explicit Messaging Connection was established across Group 2 and the Message Body Format was defined by the Server as DeviceNet (16/16).

**Figure 3-4.15 Apply\_Attributes Service Example**



The following information is specified by a Connection Object Instance within the Service Data Field of a successful Set\_Attribute\_Single response associated with the modification of the **expected\_packet\_rate** attribute.

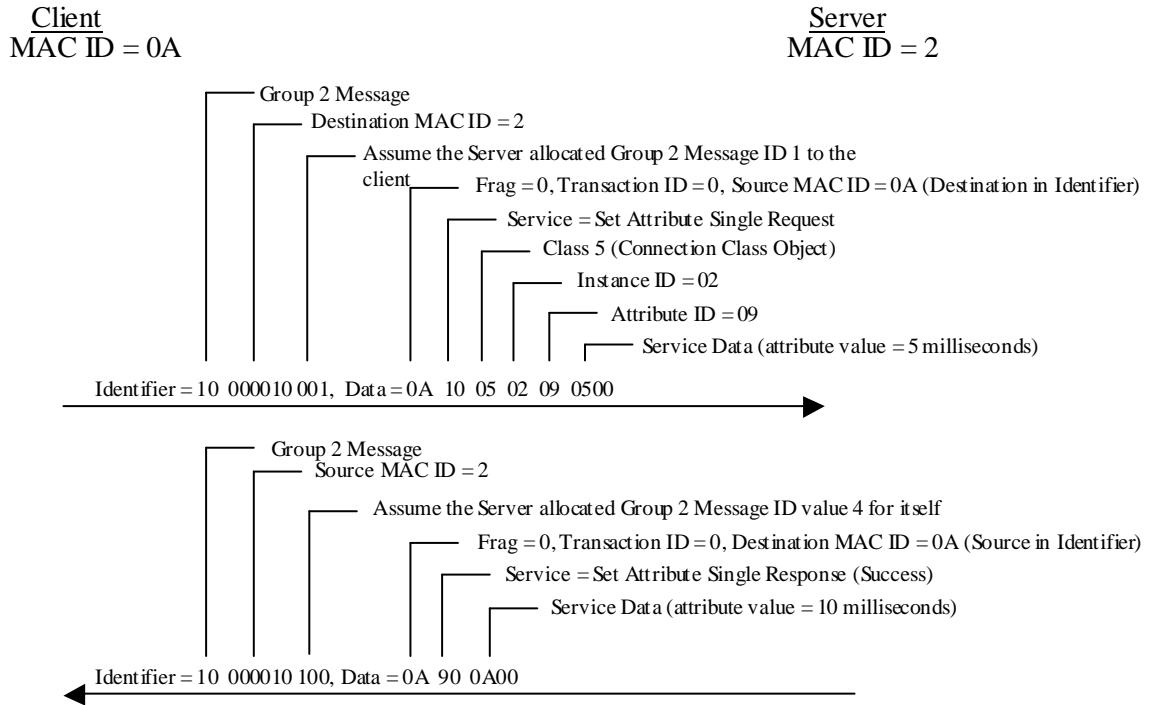
**Table 3-4.19 Service Data For Set\_Attribute\_Single [expected\_packet\_rate] Response**

Name	Data Type	Description of Parameter
Expected Packet Rate	UINT	Contains the actual value within the Connection Instance's <b>expected_packet_rate</b> attribute.

The illustration in Figure 3-4.16 depicts a Client requesting that the expected\_packet\_rate be set to 5 milliseconds. The Server returning a successful response stating that the attribute was actually set to 10 milliseconds.

For this example, assume a DeviceNet Explicit Messaging Connection was established across Group 2 and the Message body Format was defined by the Server as DeviceNet (8/8). Assume also that the expected\_packet\_rate attribute of Connection Instance #1 is requested to be set to 5 milliseconds but the product supports a 10 millisecond resolution.

**Figure 3-4.16 Set Expected Packet Rate Example**



The Connection Object does not return any information in the Service Data Field of a Set\_Attribute\_Single response directed towards any other of its attributes.

The Connection Object also supports the following internal services:

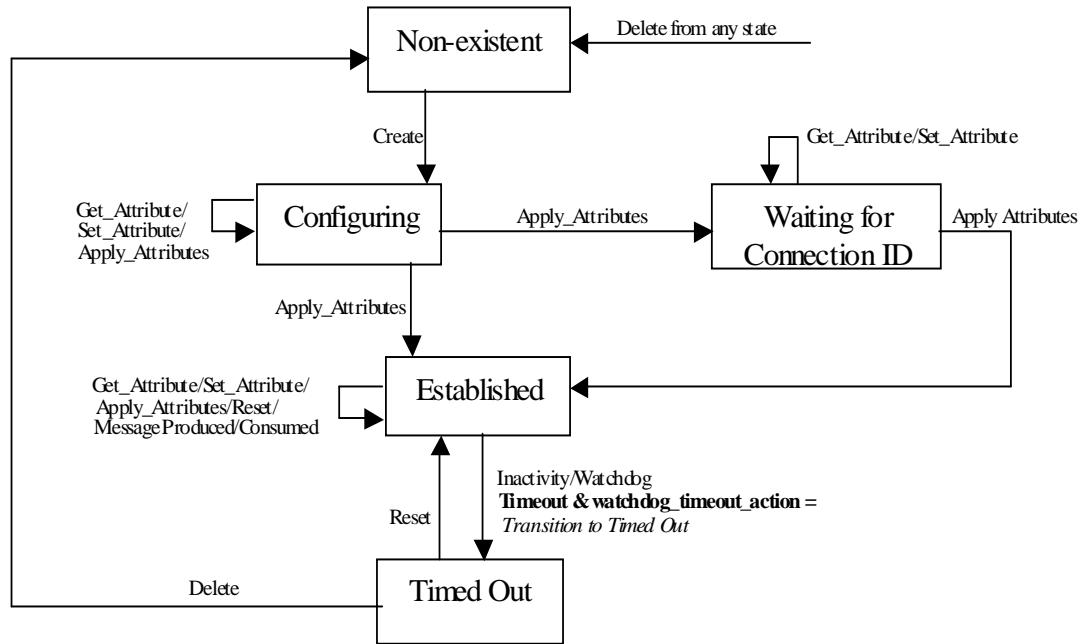
- Send\_Message - Used internally to trigger the transmission of a message. This results in the invocation of the associated Link Producer's Send service. If the message needs to be transmitted in a fragmented fashion, this service breaks up the message into multiple fragments and invokes the associated Link Producer's Send service multiple times.
- Receive\_Data - Used internally to deliver a received frame to the Connection Object. This service is invoked internally by a Link Consumer. The Connection Object is responsible for determining whether or not it must reassemble a series of data fragments into a complete message before processing the message.

## 3-4.7 Connection Instance Behavior

### 3-4.7.1 I/O Connection Instance Behavior

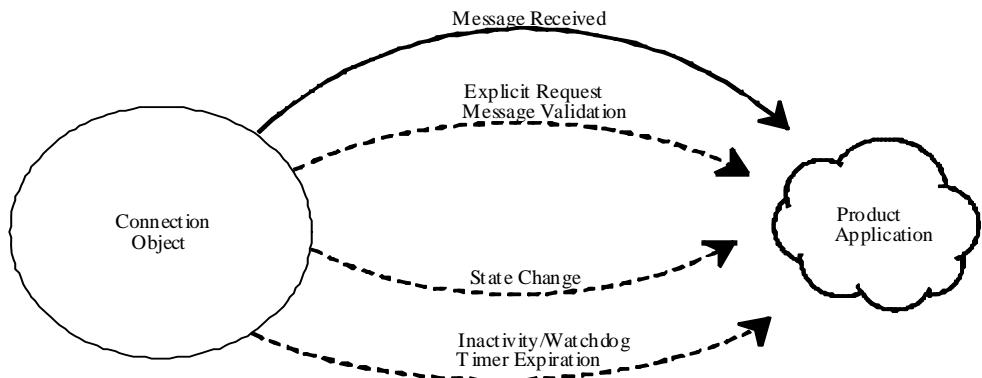
Figure 3-4.17 provides a general overview of the behavior associated with an **I/O Connection Object** (`instance_type` attribute = I/O).

Figure 3-4.17 I/O Connection Object State Transition Diagram



A Connection Object issues the internal indications described in Figure 3-4.18 to an application within a product. The purpose of this specification is to describe externally visible behaviors; rules pertinent to specific internal indications are not defined.

Figure 3-4.18 Internal Indications Issued by a Connection Object (Conceptual)



**Important:** Table 3-4.20 provides a detailed State Event Matrix for an I/O Connection Object and implementations should be based on this information. This State Event Matrix does not dictate rules with regards to product specific, internal logic. Any attempt to access the Connection Class or a Connection Object Instance may need to pass through product specific verification. This may result in an error scenario that is not indicated by the SEM in Table 3-4.20. This may also result in additional, product specific indications delivered from a Connection Object to the application and/or a specific Application Object. The point to remember is that the Connection Object must exhibit the externally visible behavior specified by the SEM and the attribute definitions (section 3-4.4).

**Table 3-4.20 I/O Connection State Event Matrix**

Event	I/O Connection Object State				
	Non-Existent	Configuring	Waiting for Connection ID	Established	Timed Out
Connection Class receives a Create Request	Class instantiates a Connection Object. Set instance_type to I/O. Set all other attributes to default values. Transition to Configuring	Not applicable	Not applicable	Not applicable	Not applicable
Connection Class receives a Delete Request	Error: Object does not exist (General Error Code 16hex)	Release all associated resources. Transition to Non-existent	Release all associated resources. Transition to Non-existent.	Release all associated resources. Transition to Non-existent.	Release all associated resources. Transition to Non-existent.
Set_Attribute_Single	Error: Object does not exist (General Error Code 16hex)	Validate/service the request based on internal logic and per the Access Rules presented in section 3-4.8. Return response.	If request to modify produced or consumed_connection_id then validate the value and service the request. Return appropriate response. If this is a request to access an attribute other than the produced or consumed_connection_id, then return an Error Response whose General Error Code is set to 0Chex (The object can not perform the requested service in its current mode/state)	Validate/service the request based on internal logic and per the Access Rules presented in section 3-4.8. Return appropriate response.	Validate/service the request based on internal logic and per the Access Rules presented in section 3-4.8. Return appropriate response.
Get_Attribute_Single	Error: Object does not exist (General Error Code 16hex)	Validate/service the request based on internal logic and per the Access Rules presented in section 3-4.8. Return response.	Validate/service the request based on internal logic and per the Access Rules presented in section 3-4.8. Return response.	Validate/service the request based on internal logic and per the Access Rules presented in section 3-4.8. Return response.	Validate/service the request based on internal logic and per the Access Rules presented in section 3-4.8. Return response.

Event	I/O Connection Object State				
	Non-Existent	Configuring	Waiting for Connection ID	Established	Timed Out
Reset	Error: Object does not exist (General Error Code 16hex)	Error: The object cannot perform the requested service in its current mode/state. (General Error Code value =)	Error: The object cannot perform the requested service in its current mode/state. (General Error Code value = 0Chex)	Cancel the current Inactivity/Watchdog Timer. Using the value in the expected_packet_rate attribute, re-start the Inactivity/Watchdog Timer. A success response is returned even if an Inactivity/Watchdog Timer is not utilized by the Connection Object (Client Transport Class 0, expected_packet_rate = 00Chex).	<p>Using the value in the expected_packet_rate attribute, start the Inactivity/Watchdog timer <u>and transition back to the Established state.</u></p> <p>If the expected_packet_rate attribute has been set to zero (0) while the Connection was in the Timed Out state, then just transition back to Established without activating an Inactivity/Watchdog Timer.</p>

Event	I/O Connection Object State				
	Non-Existent	Configuring	Waiting for Connection ID	Established	Timed Out
Apply_Attributes	Error: Object does not exist (General Error Code 16hex)	Deliver Connection Object to the application which validates the attribute information. If either of the connection_id attributes (produced or consumed) needs to be configured and cannot be generated by this module, then perform all the other steps necessary to configure the connection, return a Successful Response and transition to the Waiting For Connection ID state. The inability to generate a produced or consumed connection ID value IS NOT reported as an error. If all attributes are validly configured, then perform all the steps necessary to satisfy this connection, start all required timers, and transition to the Established state. If an error is detected, then an Error Response is returned and the Connection remains in the Configuring state <sup>1</sup> and, if a Client Connection with a production trigger value of 0 or 1 (Cyclic or Change of State), produce initial data.	If either of the connection_id attributes (produced or consumed) still needs to be configured and cannot be generated by this module, then return a Successful Response and remain in the Waiting For Connection ID state. The inability to generate a produced or consumed connection ID value IS NOT reported as an error. If the produced and/or consumed_connection_id attributes are now validly configured, then transition to the Established state and return a Successful Response1 and, if a Client Connection with a production trigger value of 0 or 1 (Cyclic or Change of State), produce initial data.	All modifications take place immediately once the Connection has transitioned to the Established state. Return Error: The object cannot perform the requested service in its current mode/state. (General Error Code value = 0Chex)	Error: The object cannot perform the requested service in its current mode/state. (General Error Code value = 0Chex)

Event	I/O Connection Object State				
	Non-Existent	Configuring	Waiting for Connection ID	Established	Timed Out
Receive_Data	Not applicable	Discard the message	Discard the message	If a complete, valid <sup>2</sup> message has been received, reset the Inactivity/Watchdog Timer <sup>3</sup> and deliver the I/O Message to the Application. <u>A Connection Object must exhibit the externally visible behavior associated with the current state of its attributes (see section 3-4.4).</u> If this is a fragmented portion of an I/O Message, process as specified by subnet.	Discard the message
Send_Message	Not applicable	Return internal error - do not send the message	Return internal error - do not send the message	Transmit the complete I/O Message or fragment as required by the subnet. If this is a Client Connection and the expected_packet_rate attribute is non-zero, restart the Transmission Trigger Timer.	Return internal error - do not send the message

Event	I/O Connection Object State				
	Non-Existent	Configuring	Waiting for Connection ID	Established	Timed Out
Inactivity/ Watchdog Timer expires	Not applicable	Not applicable	Not applicable	Examine the watchdog_timeout_action attribute of the Connection and perform the indicated action. If the watchdog_timeout_action attribute indicates that the Connection is to remain in the Established state ( <i>Auto Reset</i> ), then immediately re-start the Inactivity/Watchdog Timer.	Not applicable

- 1 If the configuration indicates that a Message ID needs to be allocated and an available Message ID does not exist in the specified Message Group, then an Error Response whose General Error Code indicates Resource Unavailable (02hex) is returned. If a Connection Object attribute value passed the range check when it was initially configured but the attribute value conflicts with another piece of information in the node when the Apply request is processed, then an Error Response is returned whose General Error Code is set to Invalid Attribute Value (09hex) and whose Additional Code is set to the Attribute ID of the offending Connection Object Attribute ID.
- 2 The Connection Object verifies that the length of the received I/O Message is less than or equal to the consumed\_connection\_size attribute prior to processing the message. If the length of the received message is less than or equal to the consumed\_connection\_size attribute, then the I/O Connection Object resets the Inactivity/Watchdog Timer, exhibits the externally visible behavior indicated by its attribute settings, and delivers the message to the Application. If the length of the received message is greater than the consumed\_connection\_size attribute, then the I/O Connection Object immediately discards the message and discontinues any subsequent processing. This is the only message content validation performed by an I/O Connection Object. Subsequent validation must be performed by the Application.
- 3 If a fragmented message is being received, then the Inactivity/Watchdog Timer is not reset until the entire message has been validly received.

**Important:** The Receive\_Data event is only delivered to an I/O Connection when a message whose Connection Identifier Field matches the consumed\_connection\_id attribute is received. If a message is received whose Connection Identifier Field does not match any Established Connection Object's consumed\_connection\_id attribute, then the message is discarded. Connection Identifiers are subnet-type specific.

If an implementation detects that it does not support an Explicit Messaging Service indicated in Table 3-4.20, then an Error Response specifying *Service Not Supported* (General Error Code 08) is returned.

### 3-4.7.2 CIP Bridged Connection Instance Behavior

Figure 3-4.19 provides a general overview of the behavior associated with a **CIP Bridged Connection Object** (`instance_type` attribute = CIP Bridged). CIP Bridged connections are used to make connections *offline*. Both I/O and Explicit Messaging can be accomplished using this connection type. The Connection Manager Object definition provides more details these types of connections.

Figure 3-4.19 CIP Bridged Connection Object State Transition Diagram

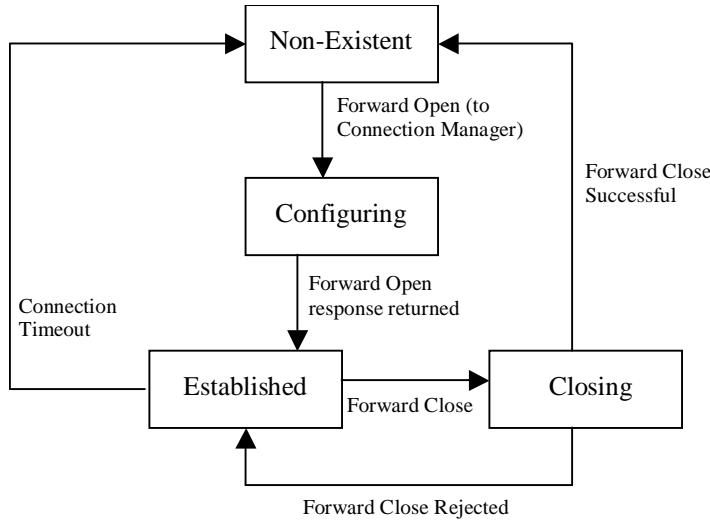


Table 3-4.21 provides a detailed State Event Matrix for a CIP Bridged Connection Object.

Table 3-4.21 CIP Bridged Connection State Event Matrix

Event	CIP Bridged Connection Object State			
	Non-Existent	Configuring	Established	Closing
Connection Manager receives a Forward Open Request	Connection Class instantiates a Connection Object. Set <b>instance_type</b> to <b>CIP Bridged</b> . Set attributes to value delivered by Forward Open service or default for CIP Bridged connections. Transition to <b>Configuring</b> .	Not applicable	Not applicable	Not applicable
Connection Manager receives notification that connection establishment is complete to target	Not applicable	Transition to <b>Established</b>	Not applicable	Not applicable
Connection Manager receives a Forward Close Request	Not applicable	Ignore event	Transition to <b>Closing</b>	Ignore event
Connection Manager receives a Forward Close Response	Not applicable	Release all resources and transition to <b>Non-Existent</b>	Release all resources and transition to <b>Non-Existent</b>	Release all resources and transition to <b>Non-Existent</b>
Delete	Error: Object does not exist (General Error Code 16 <sub>hex</sub> )	Release all resources and transition to <b>Non-Existent</b>	Release all resources and transition to <b>Non-Existent</b>	Release all resources and transition to <b>Non-Existent</b>

Event	CIP Bridged Connection Object State			
	Non-Existent	Configuring	Established	Closing
Get_Attribute_Single	Error: Object does not exist (General Error Code 16 <sub>hex</sub> )	Validate/service the request based on internal logic and per the Access Rules presented in section 3-4.8.	Validate/service the request based on internal logic and per the Access Rules presented in section 3-4.8.	Validate/service the request based on internal logic and per the Access Rules presented in section 3-4.8.
Set_Attribute_Single	Error: Object does not exist (General Error Code 16 <sub>hex</sub> )	Return appropriate response.	Return appropriate response.	Return appropriate response.
Reset	Error: Object does not exist (General Error Code 16 <sub>hex</sub> )	Error: Service Not Supported (General Error Code 08 <sub>hex</sub> )	Error: Service Not Supported (General Error Code 08 <sub>hex</sub> )	Error: Service Not Supported (General Error Code 08 <sub>hex</sub> )
Apply_Attributes	Error: Object does not exist (General Error Code 16 <sub>hex</sub> )	Error: Service Not Supported (General Error Code 08 <sub>hex</sub> )	Error: Service Not Supported (General Error Code 08 <sub>hex</sub> )	Error: Service Not Supported (General Error Code 08 <sub>hex</sub> )
Receive_Data	Not applicable	Ignore event	Invoke send service of connection object on destination port, passing the received data.	Ignore event
Send_Message	Not applicable	Ignore event	Send data on subnet.	Ignore event
Inactivity/Watchdog Timer expires	Not applicable	Not applicable	Release all resources and transition to Non-Existent	Release all resources and transition to Non-Existent

### 3-4.7.3 Explicit Messaging Connection Instance Behavior

Figure 3-4.20 provides a general overview of the behavior associated with an **Explicit Messaging Connection Object** (`instance_type` attribute = Explicit Messaging).

Figure 3-4.20 Explicit Messaging Connection Object State Transition Diagram

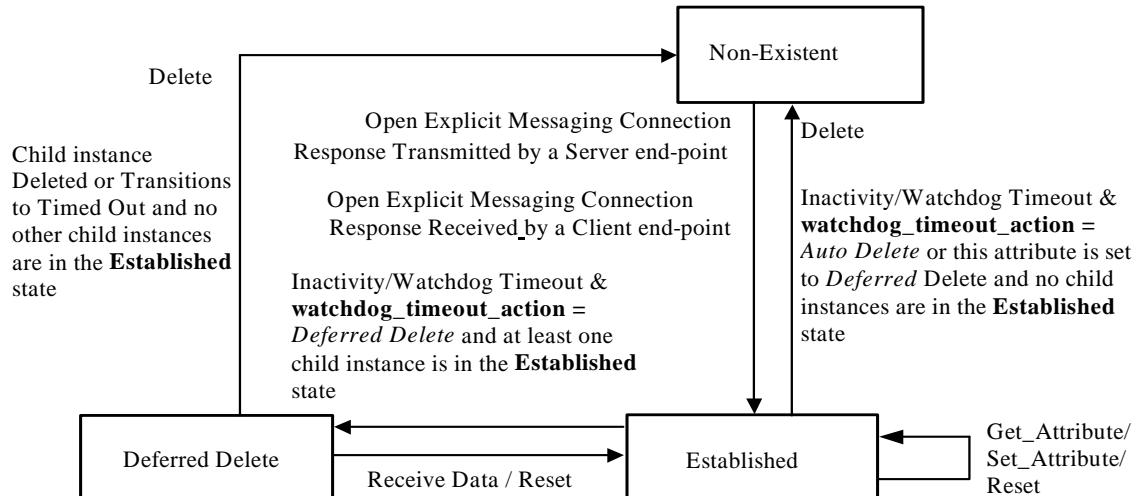


Table 3-4.22 provides a detailed State Event Matrix for an Explicit Messaging Connection Object. Implementations should be based on the information in Table 3-4.22.

**Table 3-4.22 Explicit Messaging Connection State Event Matrix**

Event	Explicit Messaging Connection Object State		
	Non-Existent	Established	Deferred Delete
UCMM receives an Open Explicit Messaging Connection Request/Response and invokes the Create service of the Connection Class	If possible, Class instantiates Connection Object. Set <b>instance_type</b> attribute to <i>Explicit Messaging</i> <sup>1</sup> . Other attributes are automatically configured using the information in the Open Explicit Messaging Connection Request/Response. Transition to <b>Established</b> . If request received, Transmit Open Explicit Messaging Connection Response.	Not applicable	Not applicable
UCMM receives a Close Request and invokes the Delete service of the Connection Class	Error: Object does not exist (General Error Code 16hex)	Release all associated resources. Transition to <b>Non-existent</b> .	Release all associated resources. Transition to <b>Non-existent</b> .
Connection Class receives a Delete Request	Error: Object does not exist (General Error Code 16hex)	Release all associated resources. Transition to <b>Non-existent</b> .	Release all associated resources. Transition to <b>Non-existent</b> .
Set_Attribute_Single	Error: Object does not exist (General Error Code 16hex)	Validate/service the request based on internal logic and per the Access Rules presented in section 3-4.8. Return appropriate response.	Validate/service the request based on internal logic and per the Access Rules presented in section 3-4.8. Return appropriate response.
Get_Attribute_Single	Error: Object does not exist (General Error Code 16hex)	Validate/service the request based on internal logic and per the Access Rules presented in section 3-4.8. Return appropriate response.	Validate/service the request based on internal logic and per the Access Rules presented in section 3-4.8. Return appropriate response.
Reset	Error: Object does not exist (General Error Code 16hex)	Cancel the current Inactivity/Watchdog Timer. Using the value in the <b>expected_packet_rate</b> attribute, re-start the Inactivity/Watchdog Timer.	Using the value in the <b>expected_packet_rate</b> attribute, re-start the Inactivity/Watchdog Timer <b>and transition back to the Established state</b> .
Apply_Attributes	Error: Object does not exist (General Error Code 16hex)	Error: The object cannot perform the requested service in its current mode/state. (General Error Code value = 0Chex)	Error: The object cannot perform the requested service in its current mode/state. (General Error Code value = 0Chex)

Event	Explicit Messaging Connection Object State		
	Non-Existent	Established	Deferred Delete
Receive_Data	Not applicable	If a valid message or message fragment has been received, then reset the Inactivity/Watchdog Timer <sup>2</sup> . Either process/store the fragment or handle the Explicit Message.	If a valid message or message fragment has been received, then restart the Inactivity/Watchdog Timer <sup>2</sup> and transition back to the <b>Established</b> state. Either process/store the fragment or handle the Explicit Message.
Send_Message	Not applicable	Examine the length of the message to transmit and, if necessary, perform a fragmented series of transmissions. Otherwise, transmit the complete Explicit Message.	Examine the length of the message to transmit and, if necessary, perform a fragmented series of transmissions. Otherwise, transmit the complete Explicit Message.
Inactivity/Watchdog Timer expires	Not applicable	If the <b>watchdog_timeout_action</b> attribute is set to <i>Auto Delete</i> or is set to <i>Deferred Delete</i> and no child connection instances are in the <b>Established</b> state release all associated resources and Transition to <b>Non-existent</b> . If the <b>watchdog_timeout_action</b> attribute is set to <i>Deferred Delete</i> and at least one child connection instance is in the <b>Established</b> state transition to <b>Deferred Delete</b>	Not applicable.
Child connection instance Deleted or Transitions to Timed Out	Not applicable	Ignore event	If no other child connection instances are in the <b>Established</b> state release all associated resources and transition to <b>Non-existent</b> .

- 1 If the configuration indicates that a connection resource needs to be allocated and an available resource does not exist, then an Error Response who's General Error Code indicates Resource Unavailable (02hex) is returned.
- 2 On DeviceNet, the MAC ID field within the Message Header of all Explicit Messages and/or Message Fragments is examined. If the Destination MAC ID is specified in the Connection ID (CAN Identifier Field), then the Source MAC ID of the other end-point must be specified in the Message Header. If the Source MAC ID is specified in the Connection ID, then the receiving module's MAC ID must be specified in the Message Header. If either of these checks fail, then the Inactivity/Watchdog Timer IS NOT reset and the message/message fragment is discarded.

**Important:** The Receive\_Data event is only delivered to an Explicit Messaging Connection when a message who's Connection ID matches the consumed\_connection\_id attribute is received. If a message is received whose connection id does not match any Established Connection Object's consumed\_connection\_id attribute, then the message is discarded.

If an implementation detects that it does not support an Explicit Messaging Service indicated in Table 3-4.22, then an Error Response specifying Service Not Supported (General Status Code 08) is returned.

### 3-4.8 Connection Object Attribute Access Rules

During the configuration of a Connection instance using the Set\_Attribute service, a module must perform value checks of each separate attribute when is modified. If an error is detected, then an Error Response is returned. The discovery of an error DOES NOT cause the deletion of the Connection instance.

**Important:** If the produced\_connection\_id and/or consumed\_connection\_id attributes contain a non-default value upon reception of the Apply Request, then the related portion of the initial\_comm\_characteristics attribute is ignored and the ID value is validated and used.

The following series of tables indicates when an attribute can be read or written via a Get and/or Set operation based on the Connection's **state** and **instance\_type**.

**Important:** If a request to Get/Set a supported attribute is received but the current state and/or instance\_type of Connection dictates that the requested access is invalid, then the returned error status indicates: *The object cannot perform the requested service in its current mode/state* (General Status Code value = 0C<sub>hex</sub> - see Appendix B for more information on status codes).

**Table 3-4.23 I/O Connection Object Attribute Access**

Attribute	I/O Connection State				
	Non-Existent	Configuring	Waiting for Connection ID	Established	Timed Out
State	Not available	Get Only	Get Only	Get Only	Get Only
instance_type	Not available	Get Only	Get Only	Get Only	Get Only
transport Class_trigger	Not available	Get/Set	Get Only	Get Only	Get Only
produced_connection_id	Not available	Get/Set	Get/Set	Get/Set	Get/Set
consumed_connection_id	Not available	Get/Set	Get/Set	Get/Set	Get/Set
initial_comm_characteristics	Not available	Get/Set	Get Only	Get Only	Get Only
produced_connection_size	Not available	Get/Set	Get Only	Get Only	Get Only
consumed_connection_size	Not available	Get/Set	Get Only	Get Only	Get Only
expected_packet_rate <sup>1</sup>	Not available	Get/Set	Get Only	Get/Set	Get/Set
watchdog_timeout_action	Not available	Get/Set	Get Only	Get/Set	Get/Set
produced_connection_path_length	Not available	Get Only	Get Only	Get Only	Get Only
produced_connection_path	Not available	Get/Set	Get Only	Get Only	Get Only
consumed_connection_path_length	Not available	Get Only	Get Only	Get Only	Get Only
consumed_connection_path	Not available	Get/Set	Get Only	Get Only	Get Only
production_inhibit_time	Not available	Get/Set	Get Only	Get/Set	Get/Set
connection_timeout_multiplier	Not available	Get/Set	Get Only	Get/Set <sup>1</sup>	Get/Set
Connection_binding_list	Not available	Get Only	Get Only	Get Only	Get Only

Attribute	I/O Connection State				
	Non-Existent	Configuring	Waiting for Connection ID	Established	Timed Out
1 When a Connection Object is in the <b>Established</b> state, any modifications to the <b>expected_packet_rate</b> or <b>connection_timeout_multiplier</b> attributes have immediate effect on the Inactivity/Watchdog Timer. The following steps are performed by a Connection Object in the <b>Established</b> state when a request is received to modify the <b>expected_packet_rate</b> or <b>connection_timeout_multiplier</b> attribute:					
	<ul style="list-style-type: none"> <li>• the current Inactivity/Watchdog Timer is canceled</li> <li>• a new Inactivity/Watchdog Timer is activated based on the new value in the <b>expected_packet_rate</b> or <b>connection_timeout_multiplier</b> attribute.</li> </ul>				

**Table 3-4.24 CIP Bridged Connection Object Attribute Access**

Attribute	Default	CIP Bridged Connection State			
	Value <sup>1</sup>	Non-Existent	Configuring	Established	Closing
state	1	Not Available	Get Only	Get Only	Get Only
instance_type	2	Not Available	Get Only	Get Only	Get Only
transportClass_trigger	From service	Not Available	Get Only	Get Only	Get Only
produced_connection_id	From service	Not Available	Get Only	Get Only	Get Only
consumed_connection_id	From service	Not Available	Get Only	Get Only	Get Only
initial_comm_characteristics	From service	Not Available	Get Only	Get Only	Get Only
produced_connection_size	From service	Not Available	Get Only	Get Only	Get Only
consumed_connection_size	From service	Not Available	Get Only	Get Only	Get Only
expected_packet_rate	From service	Not Available	Get Only	Get Only	Get Only
watchdog_timeout_action	1	Not Available	Get Only	Get Only	Get Only
produced_connection_path_length	From service	Not Available	Get Only	Get Only	Get Only
produced_connection_path	From service	Not Available	Get Only	Get Only	Get Only
consumed_connection_path_length	From service	Not Available	Get Only	Get Only	Get Only
consumed_connection_path	From service	Not Available	Get Only	Get Only	Get Only
production_inhibit_time	From service	Not Available	Get Only	Get Only	Get Only
Connection_timeout_multiplier	From service	Not Available	Get Only	Get Only	Get Only
Connection_binding_list	Empty	Not Available	Get Only	Get Only	Get Only

1 The default value is either the value indicated or is set based on one of the parameters of the Forward Open service received by the Connection Manager object.

**Table 3-4.25 Explicit Messaging Connection Object Attribute Access**

Attribute	Explicit Messaging Connection State	
	Non-Existent	Established/Deferred Delete
State	Not available	Get Only
instance_type	Not available	Get Only
transportClass_trigger	Not available	Get Only
produced_connection_id	Not available	Get Only
consumed_connection_id	Not available	Get Only
initial_comm_characteristics	Not available	Get Only
produced_connection_size	Not available	Get Only
consumed_connection_size	Not available	Get Only
expected_packet_rate <sup>1</sup>	Not available	Get/Set <sup>1</sup>
watchdog_timeout_action	Not available	Get/Set
produced_connection_path_length	Not available	Get Only
produced_connection_path	Not available	Get Only
consumed_connection_path_length	Not available	Get Only
consumed_connection_path	Not available	Get Only
production_inhibit_time	Not available	Get Only
connection_timeout_multiplier	Not available	Get/Set <sup>1</sup>
Connection_binding_list	Not available	Get Only

- 1 When a Connection Object is in the Established state, any modifications to the expected\_packet\_rate or connection\_timeout\_multiplier attributes have immediate effect on the Inactivity/Watchdog Timer. The following steps are performed by a Connection Object in the Established state when a request is received to modify the expected\_packet\_rate or connection\_timeout\_multiplier attribute:
- the current Inactivity/Watchdog Timer is canceled
  - a new Inactivity/Watchdog Timer is activated based on the new value in the expected\_packet\_rate or connection\_timeout\_multiplier attribute.

## 3-5 Connection Manager Object Class Definition

### Class Code: 06 hex

The Connection Manager Class allocates and manages the internal resources associated with both I/O and Explicit Messaging Connections. The specific instance generated by the Connection Manager Class is referred to as a Connection Instance or a Connection Object.

#### 3-5.1 Connection Manager Object Class Attributes

The Connection Manager Class attributes are defined below in Table 3-5.1.

**Table 3-5.1 Connection Manager Class Attributes**

Attribute ID	Need In Implementation	Access Rule	Attribute Name	Data Type	Attribute Description	Semantics of Values
1 thru 7	These class attributes are optional and are described in Chapter 4 of this specification.					

#### 3-5.2 Connection Manager Object Class Services

The Connection Manager Class supports the following CIP Common Services:

**Table 3-5.2 Connection Manager Class Services**

Service Code	Need In Implementation	Service Name	Service Description
01hex	Optional	Get_Attributes_All	Returns the contents of all attributes of the class.
0Ehex	Conditional <sup>1</sup>	Get_Attribute_Single	Used to read a Connection Manager Class attribute value.

<sup>1</sup> This service is Required if any of the Connection Manager Class Attributes are supported and the Get\_Attribute\_All service is not supported.

#### 3-5.2.1 Class Level Get\_Attributes\_All Response

At the **Class level**, the order of the attributes returned in the “Object/service specific reply data” portion of the Get\_Attributes\_All response is as follows:

**Table 3-5.3 Get\_Attributes\_All Response Data**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Revision (low byte) Default = 1							
1	Revision (high byte) Default = 0							
2	Max Instance (low byte) Default = 1							
3	Max Instance (high byte) Default = 0							
4	Max ID Number of Class Attributes (low byte) Default = 0							
5	Max ID Number of Class Attributes (high byte) Default = 0							
6	Max ID Number of Instance Attributes (low byte) Default = 0							
7	Max ID Number of Instance Attributes (high byte) Default = 0							

**Important:** Insert default values for all unsupported attributes.

### 3-5.3 Connection Manager Object Instance Attributes

The following list provides a summary of the Connection Manager Instance attributes and their associated data types.

**Table 3-5.4 Connection Manager Object Instance Attributes**

Attr ID	Need In Implem	Access Rule	NV	Attribute Name	Data Type	Description of Attribute	Semantics
1	Optional	Set	V	Open Requests	UINT	Number of Forward Open service requests received.	
2	Optional	Set	V	Open Format Rejects	UINT	Number of Forward Open service requests which were rejected due to bad format.	
3	Optional	Set	V	Open Resource Rejects	UINT	Number of Forward Open service requests which were rejected due to lack of resources.	
4	Optional	Set	V	Open Other Rejects	UINT	Number of Forward Open service requests which were rejected for reasons other than bad format or lack of resources.	
5	Optional	Set	V	Close Requests	UINT	Number of Forward Close service requests received.	
6	Optional	Set	V	Close Format Requests	UINT	Number of Forward Close service requests which were rejected due to bad format.	
7	Optional	Set	V	Close Other Requests	UINT	Number of Forward Close service requests which were rejected for reasons other than bad format.	
8	Optional	Set	V	Connection Timeouts	UINT	Total number of connection timeouts that have occurred in connections controlled by this Connection Manager	
9	Optional	Get	V	Connection Entry List	STRUCT of	List of Connections	If the Connection Object (Class Code 0x05) is supported, the index value into the ConnOpenBits array is the Connection Instance number minus one (the first entry in the array, index value 0, is for Connection Instance 1).
				NumConnEntries	UINT	Number of connection entries. This attribute, divided by 8 and rounded up for any remainder, gives the length of the array (in bytes) of the ConnOpenBits field of this structure.	Number of bits in the ConnOpenBits attribute.

Attr ID	Need In Implem	Access Rule	NV	Attribute Name	Data Type	Description of Attribute	Semantics
				ConnOpenBits	ARRAY of BOOL	List of connection data which may be individually queried by the Get/Search Connection Data Services. Each bit represents a possible connection.	0 = Connection Instance is Non-Existent. 1 = Connection Instance is Existent. Query for more information.
10	Reserved/Obsolete						
11	Optional	Get	V	CPU_Utilization <sup>1</sup>	UINT	CPU Utilization in tenths of a percent.	Range of 0 - 1000 representing 0 to 100%.
12	Optional	Get	V	MaxBuffSize <sup>1</sup>	UDINT	Amount of buffer space originally available.	Size in bytes
13	Optional	Get	V	BufSize Remaining <sup>1</sup>	UDINT	Amount of buffer space available at this time.	Size in bytes

1 The meaning of, and method of calculating, these values are vendor specific.

### 3-5.4 Connection Manager Object Instance Common Services

The Connection Manager Object Instance supports the following CIP Common services:

**Table 3-5.5 Connection Manager Object Instance Common Services**

Service Code	Need In Implementation	Service Name	Service Description
01hex	Optional	Get_Attributes_All	Returns the contents of all attributes of the class.
0Ehex	Conditional <sup>1</sup>	Get_Attribute_Single	Used to read a Connection Manager Object instance attribute.
10hex	Optional	Set_Attribute_Single	Used to modify a Connection Manager Object instance attribute.

1 This service is Required if any of the Connection Manager Instance Attributes are supported and the Get\_Attribute\_All service is not supported.

#### 3-5.4.1 Instance Level Get\_Attributes\_All Response

At the **Instance level**, the order of the attributes returned in the “Object/service specific reply data” portion of the Get\_Attributes\_All response is as follows. This service shall only return values of supported attributes. Therefore, a device can only return values up to the last consecutive supported attribute starting from the first attribute (Attribute 1). Values from any attributes supported beyond these must be acquired via a Get\_Attribute\_Single service.

**Table 3-5.6 Instance Level Get\_Attributes\_All Service Data**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Open Requests (low byte)							
1	Open Requests (high byte)							
2	Open Format Rejects (low byte)							
3	Open Format Rejects (high byte)							
4	Open Resource Rejects (low byte)							
5	Open Resource Rejects (high byte)							
6	Open Other Rejects (low byte)							
7	Open Other Rejects (high byte)							
8	Close Requests (low byte)							

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
9	Close Requests (high byte)							
10	Close Format Rejects (low byte)							
11	Close Format Rejects (high byte)							
12	Close Other Rejects (low byte)							
13	Close Other Rejects (high byte)							
14	Connection Timeouts (low byte)							
15	Connection Timeouts (high byte)							
16	Connection Entry List - NumConnEntries (low byte)							
17	Connection Entry List - NumConnEntries (high byte)							
18	Connection Entry List – ConnOpenBits (up to first 8 BOOLS)							
19	Connection Entry List – ConnOpenBits (up to second 8 BOOLS, if applicable)							
n	Connection Entry List – ConnOpenBits (up to n <sup>th</sup> 8 BOOLS, if applicable)							
n+1	CPU_Utilization (low byte)							
n+2	CPU_Utilization (high byte)							
n+3	MaxBuffSize (low byte)							
n+4	MaxBuffSize							
n+5	MaxBuffSize							
n+6	MaxBuffSize (high byte)							
n+7	BufSize Remaining (low byte)							
n+8	BufSize Remaining							
n+9	BufSize Remaining							
n+10	BufSize Remaining (high byte)							

**Important:** There are no default values for unsupported attributes; all values returned are for supported attributes.

### 3-5.5 Connection Manager Object Instance Object Specific Services

The Connection Manager Object Instance supports the following Object Specific services:

**Table 3-5.7 Connection Manager Object Instance Object Specific Services**

Service Code	Need In Implementation	Service Name	Service Description
4E <sub>hex</sub>	Conditional	Forward_Close	Closes a connection
52 <sub>hex</sub>	Conditional	Unconnected_Send	Unconnected Send Service. Only originating devices and devices that route between links need to implement
54 <sub>hex</sub>	Conditional	Forward_Open	Opens a connection, maximum data size is 511 bytes
56 <sub>hex</sub>	Optional	Get_Connection_Data	For diagnostics of a connection
57 <sub>hex</sub>	Optional	Search_Connection_Data	For diagnostics of a connection
59 <sub>hex</sub>	Obsolete		
5A <sub>hex</sub>	Conditional	Get_Connection_Owner	Determine the owner of a redundant connection
5B <sub>hex</sub>	Optional	Large_Forward_Open	Opens a connection, maximum data size is 65535 bytes

### 3-5.5.1 Connection Manager Object Specific Service Parameters

The object specific services of the Connection Manager Object share many of the same service parameters. These parameters are defined in this section and referenced in the object specific service definitions.

The Forward\_Open, Large\_Forward\_Open and Forward\_Close services shall be sent using the UCMM or an unbridged (local) explicit messaging connection only. They shall not be sent over a bridged explicit messaging connection. This restriction is required since each node on a bridged connection needs to receive and process these services. On subnets which support UCMM messaging, the recipient (either a target or an intermediate node) shall support these services using the UCMM and, in addition, may support these services over a local explicit messaging connection. On subnets which do not support UCMM messaging, these services shall be sent using a local explicit messaging connection. See Chapter 10 for more details on bridging and routing.

#### 3-5.5.1.1 Network Connection Parameters

The Network Connection Parameters in the Forward\_Open shall be provided as a single 16-bit word that contains the fields in the following figure:

**Table 3-5.8 Network Connection Parameters for Forward\_Open**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Redundant Owner	Connection Type	Reserved	Priority	Fixed / Variable	Connection Size (in bytes)										

The Network Connection Parameters in the Large\_Forward\_Open shall be provided as a single 32-bit word that contains the fields in the following figure:

**Table 3-5.9 Network Connection Parameters for Large\_Forward\_Open**

31	30	29	28	27	26	25	24-16	15-0
Redundant Owner	Connection Type	Reserved	Priority	Fixed / Variable	Reserved	Connection Size (in bytes)		

- *Connection Size*: The maximum size, in bytes, of the data for each direction (where applicable) of the connection. For a variable sized connection, the size shall be the maximum size of the buffer for any transfer. The actual size of the transfer for a variable connection shall be equal to or less than the size specified for the network connection. The maximum buffer size shall be dependent on the links that the connection traverses. The connection size includes the sequence count and the 32-bit real time header, if present.
- *Fixed / Variable*: With a fixed size connection, the amount of data on each transmission shall be the size specified in the *Connection Size* parameter. With a variable size connection, the amount of data on each transmission may be a variable size, up to the size specified in the *Connection Size* parameter.  
0 = Fixed  
1 = Variable
- *Priority*: The priority shall be one of:  
00 = Low Priority  
01 = High Priority  
10 = Scheduled  
11 = Urgent

- *Connection Type:*  
00 = Null (may be used to reconfigure a connection)  
01 = Multicast  
10 = Point to Point  
11 = Reserved
- *Redundant Owner*  
The redundant owner bit in the O⇒T direction shall be set (= 1) to indicate that more than one owner may be permitted to make a connection simultaneously. The bit shall be clear (= 0) to indicate an exclusive-owner, input only or listen-only connection.
- *Reserved* fields shall be set to zero

### **3-5.5.1.2 Packet Interval**

The object specific services of the Connection Manager Object share many of the same service parameters. These parameters are defined in this section and referenced in the object specific service definitions.

#### ***Requested packet interval (RPI)***

The requested packet interval shall be the requested time between packets in microseconds. The format of the RPI shall be a 32-bit integer in microseconds.

#### ***Actual packet interval (API)***

The actual packet interval shall be the actual time between packets in microseconds. The format of the API shall be a 32-bit integer in microseconds.

#### ***Usage***

The requested packet interval shall be the time between packets requested by the receiving device. The value shall be used to allocate bandwidth at each of the producing nodes. The allocation of bandwidth may have to be adjusted when the actual packet rate or actual packet interval is returned, since it is possible for the two values to differ. The path time-out value at each of the intermediate and target nodes shall also be set to the connection time-out multiplier times the API. The RPI is therefore required for all connections.

#### ***Scheduled priority***

For scheduled priority, the RPI shall be the packet rate of the repetitive data. On links that support bandwidth allocation, bandwidth shall be reserved for this packet. For scheduled priority, the data shall also be restricted to the specified packet rate, which means that if data arrives at an intermediate node faster than the specified packet rate, the node shall filter the packets to the specified rate. Since each node's scheduled priority update rate is in discrete quanta, the Actual Packet Interval (API) may be smaller (more rapid) than the RPI. The path time-out value shall be set to the Connection Time-out multiplier times the API.

### ***High priority***

For high priority, the RPI shall be used to set the path time-out in the intermediate and target nodes. The RPI shall therefore be set to the slowest packet rate expected, which shall preclude having the connection close due to a path time-out. The longer the path time-out, the longer the time required to reclaim resources in the intermediate nodes as a result of faults in the network. Since the high priority is not quantized at any of the nodes, the API shall equal the RPI. To maintain consistency, however, the time-out value shall again be set to the Connection Time-out multiplier times the API.

### ***Low priority***

For low priority, the RPI shall be used to set the path time-out in the intermediate and target nodes. The RPI shall therefore be set to the slowest packet rate expected, which shall preclude having the connection close due to a path time-out. The longer the path time-out, the longer the time required to reclaim resources in the intermediate nodes as a result of faults in the network. Since the low priority is not quantized at any of the nodes, the API shall equal the RPI. To maintain consistency, however, the time-out value shall again be set to the Connection Time-out multiplier times the API.

#### **3-5.5.1.3 Connection Timing**

The Priority/Time\_tick parameter determines the priority of the unconnected message and the time duration of a ‘tick’ (Tick Time) specified in the Time-out\_ticks parameter. The bit fields are:

**Table 3-5.10 Priority/Time\_tick Bit Definition**

7	6	5	4	3	2	1	0
Reserved		Priority 0 = Normal 1 = Reserved			Tick Time		

The Reserved and Priority fields shall be set to zero (0). The tick time shall be used in conjunction with the Time-out\_ticks parameter value to determine the total time out value. The following formula is used:

$$\text{Actual Time Out value} = 2^{\text{time\_tick}} \times \text{Time\_out\_tick}$$

If the tick time is 0000 (1 ms.), a Time-out\_ticks value of 5 would translate into 5 ms. If the tick time is 0010 (4 ms.), a Time-out\_ticks value of 5 would translate into 20 ms. The Tick Time value is enumerated in the following table.

**Table 3-5.11 Time Tick Value Enumeration**

Time Tick Value Enumeration		
Tick Time (binary)	Time per Tick	Max Time
0000	1 ms	255 ms
0001	2	510
0010	4	1020
0011	8	2040
0100	16	4080
0101	32	8160
0110	64	16,320
0111	128	32,640
1000	256	65,280
1001	512	130,560
1010	1024	261,120
1011	2048	522,240
1100	4096	1,044,480
1101	8192	2,088,960
1110	16,384	4,177,920
1111	32,768	8,355,840 ms

#### ***Time-out\_ticks***

The Time-out ticks parameter shall be used to specify the amount of time the originating application shall wait for the transaction to be completed. When used with the Time Tick portion of the Priority/Time\_tick field, a total timeout value can be calculated.

#### ***Timeout Algorithm***

The *Priority/Time\_tick* and *Time-out ticks* parameters are used to convey timeout information as the request flows through interconnecting devices. The originator states how long it will wait for a response (total round trip time), and each intermediate device (e.g. CIP Router) subtracts twice the actual amount of time between the reception of the packet and when processing of the packet is actually started (e.g. internal queuing/processing times, etc.) when forwarding the unconnected message along. If a CIP Router cannot determine specifically how much time to subtract, it shall subtract 512 milliseconds. Each intermediate device shall use this adjusted value as a timer for the resources used to manage the unconnected message and also to check to see if a timeout is imminent before forwarding the message. If a timeout is imminent, then the intermediate device returns an error response rather than just letting the originator timeout. Also, when an intermediate device times out, an error response is returned. In both cases all resources associated with that unconnected message are released. This facilitates the ability to know how far along the route the packet progressed before the timeout actually occurred, and also results in intermediate nodes not leaving resources assigned to a transaction that has already timed out.

#### **3-5.5.1.4 Connection Serial Number**

The connection serial number shall be a unique 16-bit value selected by the connection manager at the originator of the connection. The originator shall make sure that the 16-bit value is unique for the device. There shall be no other significance placed on the number by any other nodes in the connection path. The connection serial numbers shall be unique but do not have to be sequential. For example, an operator interface may have a large number of connections open at the same time, each with a unique number. The same values could be repeated at other operator interface stations. A possible implementation would be to have a connection list which points to the descriptor for each connection, and the connection serial number could be the index into the table.

#### **3-5.5.1.5 Connection Timeout Multiplier**

The Connection Timeout Multiplier specifies the multiplier applied to the RPI to obtain the connection timeout value. Devices shall stop transmitting on a connection whenever the connection times out even if the pending close has been sent. The multiplier shall be as represented by the following table:

**Table 3-5.12 Connection Timeout Multiplier Values**

Value	Multiplier
0	x4
1	x8
2	x16
3	x32
4	x64
5	x128
6	x256
7	x512
8 - 255	Reserved

#### **3-5.5.1.6 Vendor ID**

The Vendor ID utilized in conjunction with the Connection Manager is a reference to the Identity object instance #1 attribute #1 (Vendor ID).

#### **3-5.5.1.7 Originator Serial Number**

The Serial Number utilized in conjunction with the Connection Manager is a reference to the Identity object instance #1 attribute #6 (Serial Number).

#### **3-5.5.1.8 Connection Number**

The connection number shall be a 16-bit value that is assigned by the connection manager when a connection is opened. This value allows other nodes to obtain connection data from the connection manager. This number shall not be confused with the Connection Serial Number.

### **3-5.5.1.9 Connection Path Size**

The connection path size shall be the length of the connection path in 16-bit words. The length of the connection path varies during the connection process, since each node in the connection path removes the current port segment and forwards only the remaining path segments to the next node.

Appendix C, sections C-1.3 and C-1.4 specify the components of a connection path and sections C-1.5 and C-1.6 specify the hierarchy of a connection path.

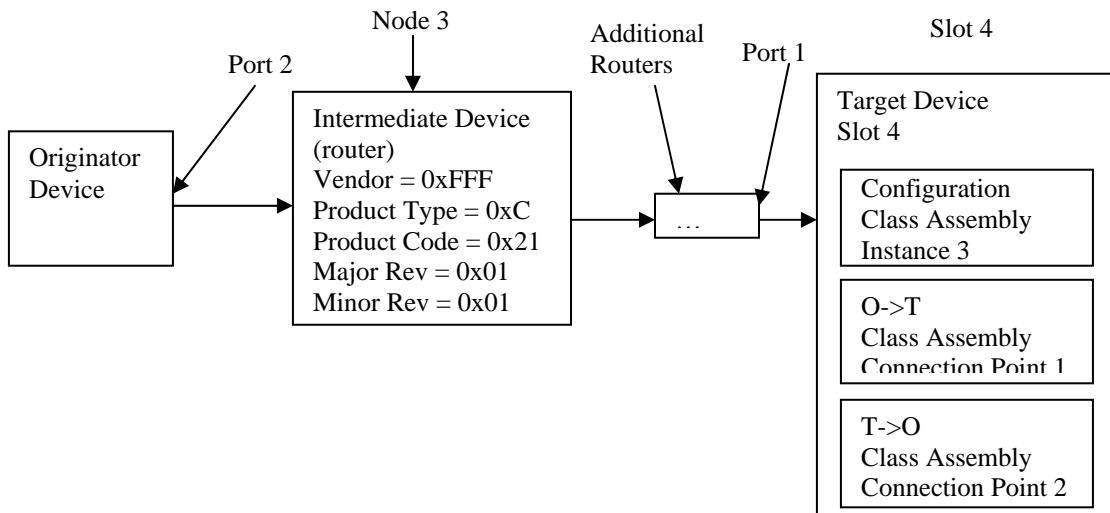
When a configuration data segment is included in the connection path the configuration data segment shall follow the applications path(s).

### **3-5.5.1.10 Connection Path**

The connection path parameter shall contain one or more encoded paths as required by a combination of the service and the value of other parameters within the service data. The format of the path(s) shall follow that as defined by the Message Router Object (Path). The first part of the connection path shall contain routing information, which may include port, network, and electronic key segments. Also, depending on the O2T\_connection\_parameters and T2O\_connection\_parameters fields and the presence of a data segment, one or more encoded application paths shall be specified. In general, the application paths are in the order of Configuration path, Consumption path, and Production path. However, a single encoded path can be used when configuration, consumption, and/or production use the same path. The following table shows the valid combinations and the implied meaning of the application paths. The application paths are relative to the *target* node.

**Table 3-5.13 Encoded Application Path Ordering**

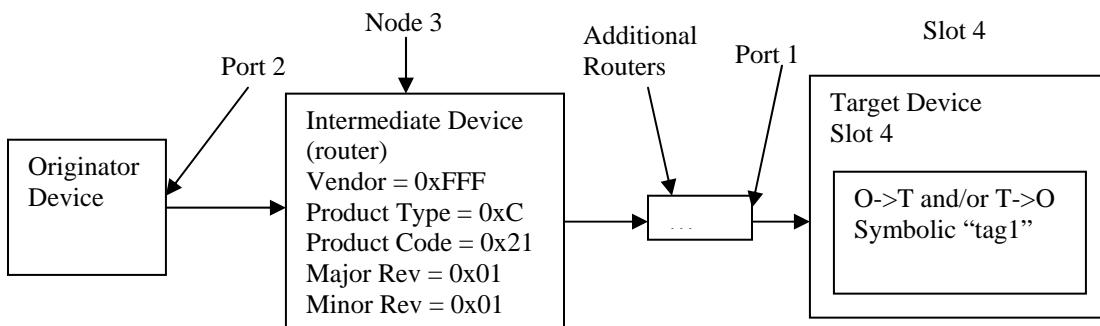
Network Connection Parameters		Data Segment Present	Number of Encoded Application Paths		
O->T Connection Type	T->O Connection Type		1	2	3
Null	Null	Yes	Path is for Configuration.	First path is for Configuration, second path is ignored	First path is for Configuration, second and third paths are ignored
		No	Invalid	Invalid	Invalid
Not Null	Null	Yes	Path is for Configuration and Consumption.	First path is for Configuration, second path is for Consumption.	Invalid
		No	Path is for Consumption.	Invalid	Invalid
Null	Not Null	Yes	Path is for Configuration and Production.	First path is for Configuration, second path is for Production.	Invalid
		No	Path is for Production.	Invalid	Invalid
Not Null	Not Null	Yes	Path is for Configuration, Consumption and Production.	First path is for Configuration, second path is for Consumption and Production	First path is for Configuration, second path is for Consumption, third path is for Production
		No	Path is for Consumption and Production.	First path is for Consumption, second path is for Production.	First path is ignored, second path is for Consumption, third path is for Production

**Figure 3-5.1 Device Diagram Showing Use of Connection Path Example in Table 3-5.14**

**Table 3-5.14 Connection Path Example Showing Logical Segments**

Segment Groups	Segments	Description
Routing	02 03	Route out port 2 to MAC ID 03
Information	34 04	Key segment
	FF FF	Vendor
	0C 00	Product Type
	21 00	Product Code
	81 01	Major revision 1, Minor Revision 1, compatible keying
	...	The routing information may continue for additional routers
	01 04	Route out port 1 (backplane) to slot 4
	43 xx	Production inhibit segment
Application Paths	20 04	Assembly object class
	24 03	Instance 3 for configuration
	2C 01	Optional O->T Connection point, specified if O->T connection type is not NULL
	2C 02	Optional O->T Connection point, specified if O->T connection type is not NULL
	80 01	Data Segment
	00 01	Configuration Data

This example includes 3 application paths, 20 04 24 03 is the configuration path, 20 04 2C 01 is the O->T path and 20 04 2C 02 is the T->O path (see Table 3-5.14).

**Figure 3-5.2 Device Diagram Showing Use of Connection Path Example in Table 3-5.15.**

**Table 3-5.15 Connection Path Example Showing a Symbolic Segment**

<b>Segment Groups</b>	<b>Segment</b>	<b>Description</b>
Routing	02 03	Route out port 2 to MAC ID 03
Information	34 04	Key segment
	FF FF	Vendor
	0C 00	Product Type
	21 00	Product Code
	01 01	Major revision 1, Minor Revision 1, exact match
	...	The routing information may continue for additional routers
	01 04	Route out port 1 (backplane) to slot 4
Application Paths	91 04	4 byte Symbol segment
	74 61	t a
	67 31	g 1

This example includes one application path, a symbolic “tag1”. At least one of the O->T and T->O network parameters connection types must not be NULL for this connection path to be valid (because there is no data segment present). If both the O->T and T->O network parameters connection types are not NULL then the both O->T and T->O application paths are symbolic “tag1” (see Table 3-5.15).

### 3-5.5.1.10.1 Connection Path Compression

When a Data Segment is present (and thus a Configuration Path), the Connection Path encoding can be compressed if all of the following are true:

- A Data Segment is present
- The Configuration Path consists of Class and Instance only
- There exists only one target Class for all encoded paths (Configuration and I/O)
- The target Class is not the Assembly Object.
- The encoded target I/O Connection Path(s) contain Connection Points

The compression consists of eliminating the need to replicate the instance value for each of the two or three paths. The Class and Instance values are applied to the Configuration Path, and the one or two Connection Points that follow are applied to the I/O Connection Path(s).

An encoded path with the contents 20 ww 24 xx 2C yy 2C zz followed by a Data Segment is decoded as follows (given that ww does not equal 04):

- 20 ww 24 xx Configuration Path (meaning of the Data Segment is defined by the object)
- 20 ww 24 xx 2C yy Consuming I/O Path
- 20 ww 24 xx 2C zz Producing I/O Path

### **3-5.5.1.10.2 Assumed Assembly Object Attribute**

When the Connection Path parameter finds a Configuration or I/O path consisting of an instance of the Assembly object (Class Code 4) without an attribute, the Data attribute (Attribute 3) is assumed.

An encoded path with the contents 20 04 24 xx 24 yy 24 zz followed by a Data Segment is decoded as follows:

- 20 04 24 xx 30 03 Configuration Path
- 20 04 24 yy 30 03 Consuming I/O Path
- 20 04 24 zz 30 03 Producing I/O Path

### **3-5.5.1.11 Network Connection ID**

The Network Connection ID shall be link specific and shall not be related to the connection serial number, which is connection specific and the same over all the links. The fields of the Network Connection ID shall be used to set the screening mechanism for the specified link. The Network Connection ID is either a CIP Produced Connection ID or CIP Consumed Connection ID.

A multicast CID shall not be reused until all connections associated with the CID have been closed or timed out.

### **3-5.5.1.12 Transport Class and Trigger**

The transport class and trigger specify the type of transport required for the connection. This information shall not be used by the connection manager but passed on to the application. The application shall determine if the transport type is supported and if an instance of the required transport is available. See the *transportClass* trigger attribute of the Connection Object for the details of this parameter.

## **3-5.5.2 Forward\_Open and Large\_Forward\_Open**

The Forward Open Service (Service Code = 54<sub>hex</sub>) and Large\_Forward\_Open service (Service Code = 5B<sub>hex</sub>) are used to establish a Connection with a Target Device<sup>1</sup>. These services result in local connection establishment on each link along the path. The Large\_Forward\_Open service differs from the Forward\_Open service only in the maximum size connection that can be established. This difference is evidenced in the data type and bit-field assignments of the O to T and T to O Network Connection parameters. All references to the Forward\_Open service in this specification shall apply to both services.

The Forward Open request sets up network, transport, and application connections. An application connection consists of a single transport connection, and one or two network connections that are in turn comprised of one or more multiple link connections. Each port segment in the connection path uses a link connection. The Forward\_Open service between two devices builds one or two link connections as specified by the network connection parameter and the requested packet intervals (RPI). Since up to two network connections can be required for a single transport connection, they are differentiated by the O->T and T->O designations; O->T means originator to target, and T->O means target to originator.

---

<sup>1</sup> The processing of a single Forward\_Open service may result in the creation of multiple active connection object instances.

A connection established to the Message Router object of the target (based on the Connection Path parameter in the Forward\_Open request) results in an Explicit Messaging type connection. The format of the data sent on this connection uses the Message Router Request/Response Format as defined in Chapter 2-4 of this specification. Connections established to any other application object result in an I/O type connection. See the Connection Object attribute 2 (Instance Type).

**Table 3-5.16 Forward\_Open / Large\_Forward\_Open Request**

Parameter Name	Data Type	Description
Priority/Time_tick	BYTE	Used to calculate request timeout information. See 3-5.5.1.3.
Time-out_ticks	USINT	Used to calculate request timeout information. See 3-5.5.1.3.
O->T Network Connection ID	UDINT	Network Connection ID to be used for the local link, originator to target. This is the originator's CIP Produced Connection ID.
T->O Network Connection ID	UDINT	Network Connection ID to be used for the local link, target to originator. This is the originator's CIP Consumed Connection ID.
Connection Serial Number	UINT	See <b>Object Specific Service Parameters</b> above.
Originator Vendor ID	UINT	Vendor ID of the originating node.
Originator Serial Number	UDINT	Serial Number of the originating node.
Connection Timeout Multiplier	USINT	See <b>Object Specific Service Parameters</b> above.
Reserved	octet	Reserved
	octet	
	octet	
O->T RPI	UDINT	Originator to Target requested packet rate, in microseconds.
O->T Network Connection Parameters	WORD/ DWORD <sup>1</sup>	See <b>Object Specific Service Parameters</b> above.
T->O RPI	UDINT	Target to Originator requested packet rate, in microseconds.
T->O Network Connection Parameters	WORD/ DWORD <sup>1</sup>	See <b>Object Specific Service Parameters</b> above.
Transport Type/Trigger	BYTE	See <b>Object Specific Service Parameters</b> above.
Connection_Path_Size	USINT	The number of 16 bit words in the <i>Connection_Path</i> field.
Connection_Path	Padded EPATH	Indicates the route to the Remote Target Device.

1 The data type of this parameter is a WORD for the Forward\_Open service, and a DWORD for the Large\_Forward\_Open service.

Success shall be returned when the connection requested has been established from this point forward in the path. This reply also shall indicate the connection serial number and the actual packet rate of the connection. Once the successful reply has been received, the connection shall be open from this point forward in the path. Targets shall wait at least 10 seconds after sending a successful response for the first packet on a connection.

**Table 3-5.17 Successful Forward\_Open / Large\_Forward\_Open Response**

Parameter Name	Data Type	Description
O->T Network Connection ID	UDINT	Network Connection ID to be used for the local link, originator to target. If chosen by the originator, then the value is echoed back. This is the target's CIP Consumed Connection ID.
T->O Network Connection ID	UDINT	Network Connection ID to be used for the local link, target to originator. If chosen by the originator, then the value is echoed back. This is the target's CIP Produced Connection ID.
Connection Serial Number	UINT	Returns same value received in the Request packet.
Originator Vendor ID	UINT	Returns same value received in the Request packet.
Originator Serial Number	UDINT	Returns same value received in the Request packet.
O->T API	UDINT	Actual packet rate, originator to target. A router shall use the lesser of this value and the T->O API for the expected_packet_rate of the connection.
T->O API	UDINT	Actual packet rate, target to originator. A router shall use the lesser of this value and the O->T API for the expected_packet_rate of the connection.
Application Reply Size	USINT	Number of 16 bit words in the <i>Application Reply</i> field.
Reserved	USINT	Reserved
Application Reply	Array of Byte	Application specific data

The following format shall be used for all Forward\_Open failures. The requested connection shall not be established, and the object specific status words shall contain information about the reason for the failure. The remaining\_path\_size shall contain the length of the path at the point the connection request failed.

In the failure response, the remaining remaining\_path\_size shall be the “pre-stripped” size. This shall be the size of the path when the node first receives the request and has not yet started processing it. A target node may return either the “pre-stripped” size or 0 for the remaining remaining\_path\_size.

A duplicate Forward\_Open service shall be defined as a Forward\_Open service whose vendor\_ID, connection\_serial\_number, and originator\_serial\_number match an existing connection’s parameters. If the duplicate Forward\_Open service is a null Forward\_Open service (defined as the connection type in both the O->T and T->O network connection parameter fields are NULL), then the Forward\_Open service shall be forwarded to the application for further processing. Null Forward\_Open requests may be used to reconfigure the connection. The Connection Manager in the intermediate nodes need not allocate additional resources for a duplicate Forward\_Open request since the resources have already been allocated. If the duplicate Forward\_Open request is not NULL, then a general status = 0x01, extended status = 0x0100 shall be returned.

**Table 3-5.18 Unsuccessful Forward\_Open / Large\_Forward\_Open Response**

Parameter Name	Data Type	Description
Connection Serial Number	UINT	Returns same value received in the Request packet.
Originator Vendor ID	UINT	Returns same value received in the Request packet.
Originator Serial Number	UDINT	Returns same value received in the Request packet.
Remaining Path Size	USINT	This field is only present with routing type errors and indicates the number of words in the original route path ( <i>Connection_Path</i> parameter of the Forward_Open Request) as seen by the router that detects the error.
Reserved	USINT	Shall be set to zero.

Either a target device or an intermediate router shall return an Invalid Connection Size extended error code, along with the maximum connection size supported additional status word, if the connection size requested is not supported while processing a Large Forward\_Open request.

The suggested originator behavior in the duplicate forward open case should be to either close and re-establish the connection or wait for the connection to time-out and then establish the connection again. It shall be recognised that in the latter case, it shall take the connection 60 seconds (first data time-out) to time-out before the connection can be re-established.

### 3-5.5.3 Forward\_Close

The Forward\_Close Service (Service Code = 4E<sub>hex</sub>) is used to close a connection with a Target Device (and all other nodes in the connection path). The Forward\_Close request shall remove a connection from all the nodes participating in the original connection. The Forward\_Close shall be sent between Connection Managers as specified in the connection\_path. The Forward\_Close request shall cause all resources in all nodes participating in the connection to be deallocated, including connection IDs, bandwidth, and internal memory buffers.

If an intermediate node cannot find the connection that is to be closed (it may have timed out at the node), the Forward\_Close request shall still be forwarded to downstream nodes or the target application.

**Table 3-5.19 Forward\_Close Service Request**

Parameter Name	Data Type	Description
Priority/Time_tick	BYTE	Used to calculate request timeout information.
Time-out_ticks	USINT	Used to calculate request timeout information.
Connection Serial Number	UINT	Connection Serial Number of established connection.
Originator Vendor ID	UINT	Vendor ID of the originating node.
Originator Serial Number	UDINT	Serial Number of the originating node.
Connection_Path_Size	USINT	The number of 16 bit words in the <i>Connection_Path</i> field.
Reserved	USINT	Reserved
Connection_Path	Padded EPATH	Indicates the route to the Remote Target Device.

Forward\_Close service request shall be successful when a request is received whose Originator Vendor ID, Connection Serial Number, and Originator Serial Number match an existing connection's parameters. Additional information provided by this service (ie, Connection Path) shall be ignored by the target.

Success shall be returned when the connection has been deleted at the target. The originator, and each intermediate node along the path, closes the connection and releases resources associated with that connection when the success response is received.

**Table 3-5.20 Successful Forward\_Close Response**

Parameter Name	Data Type	Description
Connection Serial Number	UINT	Returns same value received in the Request packet.
Originator Vendor ID	UINT	Returns same value received in the Request packet.
Originator Serial Number	UDINT	Returns same value received in the Request packet.
Application Reply Size	USINT	Number of 16 bit words in the <i>Application Reply</i> field.
Reserved	USINT	Reserved
Application Reply	Array of byte	Application specific data

**Table 3-5.21 Unsuccessful Forward\_Close Response**

Parameter Name	Data Type	Description
Connection Serial Number	UINT	Returns same value received in the Request packet.
Originator Vendor ID	UINT	Returns same value received in the Request packet.
Originator Serial Number	UDINT	Returns same value received in the Request packet.
Remaining Path Size	USINT	
Reserved	USINT	

#### **3-5.4 Unconnected\_Send**

The Unconnected\_Send service shall allow an application to send a message to a device without first setting up a connection. The Unconnected\_Send service shall use the Connection Manager object in each intermediate node to forward the message and to remember the return path. The UCMM of each link shall be used to forward the request from Connection Manager to Connection Manager just as it is for the Forward\_Open service; however, no connection shall be built. The Unconnected\_Send service shall be sent to the local Connection Manager and shall be sent between intermediate nodes. When an intermediate node removes the last port segment, the message shall be formatted as a UCMM message and sent to the port and link address of the last segment.

NOTE: The target node never sees the Unconnected\_Send service but only a standard message arriving via the UCMM.

**Table 3-5.22 Unconnected\_Send Service Parameters**

Parameter Name	Data Type	Description
Priority/Time_tick	BYTE	Used to calculate request timeout information.
Time-out_ticks	USINT	Used to calculate request timeout information.
Message_Request_Size	UINT	Specifies the number of bytes in the Message Request.
Message_Request <sup>1</sup>	Struct of	
Service	USINT	Service code of the request.
	USINT	The number of 16 bit words in the Request_Path field (next element).
	Padded EPATH	This is an array of bytes whose contents convey the path of the request (Class ID, Instance ID, etc.) for this transaction.
	Array of octet	Service specific data to be delivered in the Explicit Messaging Request. If no additional data is to be sent with the Explicit Messaging Request, then this array will be empty.
Pad	USINT	Only present if Message_Request_Size is an odd value.
Route_Path_Size	USINT	The number of 16 bit words in the Route_Path field.
Reserved	USINT	Reserved byte. Shall be set to zero (0).
Route_Path	Padded EPATH	Indicates the route to the Remote Target Device.

1 This is the Message Router Request Format as defined in Chapter 2.

The Unconnected\_Send reply shall be generated by the last intermediate node from the UCMM reply generated by the target node or by an intermediate node as the result of a UCMM time-out, a problem with the embedded message, or a problem with the Unconnected Service Request itself. The packet shall be routed from intermediate node to intermediate node using the information stored when the Unconnected\_Send request was processed. The reply shall contain a header with status information about the request and a variable length reply generated by the target node.

The application reply shall be the format of the Message reply from the target and shall contain error codes that resulted from the execution of the message by the application at the target node.

The Reply Service code returned may be either the service code sent inside the Unconnected\_Send service data *or* the Unconnected\_Send service code. In either case, the upper bit is set to indicate a response. For example, if the requested service was a Get\_Attribute\_Single, then the response message may specify either 0x8E (Get\_Attribute\_Single = 0x0E) OR 0xD2 (Unconnected\_Send = 0x52). This assists a router by:

- Allowing it to pass the original service code response back for a message which made it to the target (since the Unconnected\_Send does not appear at the target)
- Not requiring it to parse the service code out of the Unconnected\_Send request when a failure occurs in the routing.

The Service Data associated with a successful Unconnected\_Send response is:

**Table 3-5.23 Successful Unconnected\_Send Response**

Parameter Name	Data Type	Description
Reply Service	USINT	Reply service code.
Reserved	octet	Shall be zero
General Status	USINT	This value is zero (0) for successful transactions.
Reserved	USINT	Shall be zero.
Service Response Data	Array of byte	This field contains the Explicit Messaging Service Data returned by the Target Device/Object. For example, this would contain Attribute Data in response to a Get_Attribute_Single request. If the Explicit Messaging response returned by the Target Device/Object did not contain any Service Data, then this field shall be empty.

The response Service Data associated with an unsuccessful Unconnected\_Send response is defined below.

**Table 3-5.24 Unsuccessful Unconnected\_Send Response**

Parameter Name	Data Type	Description
Reply Service	USINT	Reply service code.
Reserved	USINT	Shall be zero.
General Status	USINT	One of the General Status codes listed in Appendix B Error Codes. If a routing error occurred, it shall be limited to the values specified in the Routing Error Values table.
Size of Additional Status	USINT	Number of 16 bit words in Additional Status array.
Additional Status	Array of UINT	When returning an error from a target which is a DeviceNet node, the Additional Status shall contain the 8 bit Additional Error Code from the target in the lower 8 bits and a zero (0) in the upper 8 bits.
Remaining Path Size	USINT	This field is only present with routing type errors and indicates the number of words in the original route path ( <i>Route_Path</i> parameter of the Unconnected Send Request) as seen by the router that detects the error.

In order to standardize Routing Error handling in CIP Routers, the following table presents the possible Routing Errors and when/why they would be returned. In each of these cases the *Remaining Path Size* parameter is present.

**Table 3-5.25 Routing Error Values**

<b>General Status</b>	<b>Additional Status</b>	<b>Usage</b>
01	$0204_{hex}$	Timeout indicator. Returned under the following circumstances: Failure to establish an Explicit Messaging Connection. Timeout event occurs while waiting for an Explicit Messaging Response. After decreasing the timing parameters when an Unconnected Send request is received, the CIP Router determines that there is not enough time left to continue this transaction (a Requesting Device timeout is imminent).
01	$0311_{hex}$	Invalid Port ID specified in the <i>Route_Path</i> field.
01	$0312_{hex}$	Invalid Node Address specified in the <i>Route_Path</i> field.
01	$0315_{hex}$	Invalid segment type in the <i>Route_Path</i> field.
02	empty	Resource error. The CIP Router lacks the resources to fully process the Unconnected Send Request.
04	empty	Segment type error. Indicates the CIP Router experienced a parsing error when extracting the Explicit Messaging Request from the Unconnected Send Request Service Data.

**3-5.5.5 Get\_Connection\_Data**

This service shall return the parameters associated with a specified connection\_number. The connection\_number may be different from device to device even for the same connection. The connection\_number corresponds to the offset into the Connection Manager attribute that enumerates the status of the connections.

**Table 3-5.26 Get\_Connection\_Data Service Request**

<b>Parameter Name</b>	<b>Data Type</b>	<b>Description</b>
Connection Number	UINT	Connection number

**Table 3-5.27 Get\_Connection\_Data Service Response**

<b>Parameter Name</b>	<b>Data Type</b>	<b>Description</b>
Connection Number	UINT	
Connection State	UINT	
Originator Port	UINT	
Target Port	UINT	
Connection Serial Number	UINT	Connection Serial Number of established connection.
Originator Vendor ID	UINT	Vendor ID of the originating node.
Originator Serial Number	UDINT	Serial Number of the originating node.
Originator O->T CID	UDINT	
Target O->T CID	UDINT	
Connection Timeout Multiplier	USINT	
Reserved	USINT USINT USINT	
Originator RPI O->T	UDINT	

Parameter Name	Data Type	Description
Originator API O->T	UDINT	
Originator T->O CID	UDINT	
Target T->O CID	UDINT	
Connection Timeout Multiplier	USINT	
Reserved	USINT USINT USINT	
Originator RPI T->O	UDINT	
Originator API T->O	UDINT	

### 3-5.5.6 Search\_Connection\_Data

This service shall return the parameters associated with the specified connection serial number, originator vendor ID and originator serial number.

**Table 3-5.28 Search\_Connection\_Data Service Request**

Parameter Name	Data Type	Description
Connection Serial Number	UINT	Connection Serial Number of established connection.
Originator Vendor ID	UINT	Vendor ID of the originating node.
Originator Serial Number	UDINT	Serial Number of the originating node.

The format of the Search\_Connection\_Data response shall be the same as the response from the Get\_Connection\_Data service.

## 3-5.6 Connection Manager Object Instance Error Codes

### 3-5.6.1 Error Code Listing

The error codes are returned with the reply to a Connection Manager Service Request that resulted in an error. These error codes shall be used to help diagnose the problem with a Service Request. The error code shall be split into an 8 bit general status and one or more 16-bit words of extended status. Unless specified otherwise, only the first word of extended status shall be required. Additional words of extended status may be used to specify additional module specific debug information. All devices that originate messages shall be able to handle multiple words of extended status.

The following table provides a summary of the available error codes.

**Table 3-5.29 Connection Manager Service Request Error Codes**

<b>General Status</b>	<b>Extended Status</b>	<b>Explanation and Description</b>
0x00		Service completed successfully
0x01	0x0000 through 0x00FF	Obsolete.
0x01	0x0100	<b>CONNECTION IN USE OR DUPLICATE FORWARD OPEN</b>  This extended status code shall be returned when an originator is trying to make a connection to a target with which the originator may have already established a connection (duplicate Forward_Open – see 3-5.5.2).
0x01	0x0101 through 0x0102	Reserved by CIP
0x01	0x0103	<b>TRANSPORT CLASS AND TRIGGER COMBINATION NOT SUPPORTED</b>  A transport class and trigger combination has been specified which is not supported by the target. Routers shall not fail the connection based on the transport class and trigger combination. Only targets shall return this extended status code.
0x01	0x0104 through 0x0105	Reserved by CIP
0x01	0x0106	<b>OWNERSHIP CONFLICT</b>  The connection cannot be established since another connection has exclusively allocated some of the resources required for this connection. An example of this would be that only one exclusive owner connection can control an output point on an I/O Module. If a second exclusive owner connection (or redundant owner connection) is attempted, this error shall be returned. This extended status code shall only be returned by a target node.
0x01	0x0107	<b>TARGET CONNECTION NOT FOUND</b>  This extended status code shall be returned in response to the forward_close request, when the connection that is to be closed is not found at the target node. This extended status code shall only be returned by a target node. Routers shall not generate this extended status code. If the specified connection is not found at the intermediate node, the close request shall still be forwarded using the path specified in the Forward_Close request.
0x01	0x0108	<b>INVALID NETWORK CONNECTION PARAMETER</b>  This extended status code shall be returned as the result of specifying a connection type, connection priority, redundant owner or fixed / variable that is not supported by the target application. Only a target node shall return this extended status code.
0x01	0x0109	<b>INVALID CONNECTION SIZE</b>  This extended status code is returned when the target or router does not support the specified connection size. This could occur at a target because the size does not match the required size for a fixed size connection. It could occur at a router if the requested size is too large for the specified network.  An additional status may follow indicating the maximum connection size supported by the responding node. The additional status word is required when issued in response to the Large_Forward_Open.

General Status	Extended Status	Explanation and Description
0x01	0x010A through 0x010F	Reserved by CIP
0x01	0x0110	<p>TARGET FOR CONNECTION NOT CONFIGURED</p> <p>This extended status code shall be returned when a connection is requested to a target application that has not been configured and the connection request does not contain a data segment for configuration. Only a target node shall return this extended status code.</p> <p>Reference section C-1.4.5 – DATA SEGMENT</p>
0x01	0x0111	<p>RPI NOT SUPPORTED.</p> <p>This extended status code shall be returned if the device can not support the requested O-&gt;T or T-&gt;O RPI. This extended status code shall also be used if the connection time-out multiplier produces a time-out value that is not supported by the device or the production inhibit time is not valid.</p>
0x01	0x0112	Reserved by CIP
0x01	0x0113	<p>OUT OF CONNECTIONS</p> <p>Connection Manager cannot support any more connections. The maximum number of connections supported by the Connection Manager has <u>already been created</u>.</p>
0x01	0x0114	<p>VENDOR ID OR PRODUCT CODE MISMATCH</p> <p>The Product Code or Vendor Id specified in the electronic key logical segment does not match the Product Code or Vendor Id of the target device.</p>
0x01	0x0115	<p>PRODUCT TYPE MISMATCH</p> <p>The Product Type specified in the electronic key logical segment does not match the Product Type of the target device.</p>
0x01	0x0116	<p>REVISION MISMATCH</p> <p>The major and minor revision specified in the electronic key logical segment does not correspond to a valid revision <u>of the target</u> device.</p>
0x01	0x0117	<p>INVALID PRODUCED OR CONSUMED APPLICATION PATH</p> <p>The produced or consumed application path specified in the connection path does not correspond to a valid produced or consumed application path within the target application. This error could also be returned if a produced or consumed application path was required, but not provided by a connection request.</p>
0x01	0x0118	<p>INVALID OR INCONSISTENT CONFIGURATION APPLICATION PATH</p> <p>An application path specified for the <b>configuration data</b> does not correspond to a <b>configuration application</b> or is inconsistent with the consumed or produced application paths. For example the connection path specifies <b>float configuration data</b> while the <b>produced or consumed paths</b> specify <b>integer data</b>.</p>

General Status	Extended Status	Explanation and Description
0x01	0x0119	<p>NON-LISTEN ONLY CONNECTION NOT OPENED</p> <p>Connection request fails since there are no non-listen only connection types currently open. Refer to Section 3.6 for a description of application connection types.</p> <p>The extended status code shall be returned when an attempt is made to establish a listen only connection type to a target, which has no non-listen only connection already established.</p>
0x01	0x011A	<p>TARGET OBJECT OUT OF CONNECTIONS</p> <p>The maximum number of connections supported by this instance of the target object has been exceeded.</p> <p>For example, the Connection Manager could support 20 connections while the target object can only support 10 connections. On the 11th Connection Request to the target object, this extended status code would be used to signify that the maximum number of connections already exist to the target object.</p>
0x01	0x011B	RPI IS SMALLER THAN THE PRODUCTION INHIBIT TIME
0x01	0x011C through 0x0202	Reserved by CIP
0x01	0x0203	<p>CONNECTION TIMED OUT</p> <p>This extended status code shall occur when a client tries to send a connected message over a connection that has been timed-out. This extended status code shall only occur locally at the producing node.</p>
0x01	0x0204	<p>UNCONNECTED REQUEST TIMED OUT</p> <p>The Unconnected Request Timed Out error shall occur when the UCMM times out before a reply is received. This may occur for an Unconnected_Send, Forward_Open, or Forward_Close service. This typically means that the UCMM has tried a link specific number of times using a link specific retry timer and has not received an acknowledgement or reply. This may be the result of congestion at the destination node or may be the result of a node not being powered up or present. This extended status code shall be returned by the originating node or any intermediate node.</p>
0x01	0x0205	<p>PARAMETER ERROR IN UNCONNECTED REQUEST SERVICE</p> <p>For example, this shall be caused by a Connection Tick Time (see section 3-5.5.1.3) and Connection time-out combination in an Unconnected_Send, Forward_Open, or Forward_Close service that is not supported by an intermediate node.</p>
0x01	0x0206	<p>MESSAGE TOO LARGE FOR UNCONNECTED_SEND SERVICE</p> <p>This shall be caused when the Unconnected_Send is too large to be sent out on a network.</p>
0x01	0x0207	UNCONNECTED ACKNOWLEDGE WITHOUT REPLY

General Status	Extended Status	Explanation and Description
		The message was sent via the unconnected message service and an acknowledge was received but a data response message was not received.
0x01	0x0208 through 0x0300	Reserved by CIP
0x01	0x0301	<b>NO BUFFER MEMORY AVAILABLE</b>  The extended status code shall occur when insufficient connection buffer memory is available in the target or any router devices. Routers and target nodes shall return this error.
0x01	0x0302	<b>NETWORK BANDWIDTH NOT AVAILABLE FOR DATA</b>  This extended status code shall be returned by any device in the path that is a producer and can not allocate sufficient bandwidth for the connection on its link. This can occur at any node. This can only occur for connections that are specified as scheduled priority.
0x01	0x0303	<b>NO CONSUMED CONNECTION ID FILTER AVAILABLE</b>  Any device in the path that contains a link consumer for the connection and does not have an available consumed_connection_id filter available shall return this extended status code.
0x01	0x0304	<b>NOT CONFIGURED TO SEND SCHEDULED PRIORITY DATA</b>  If requested to make a connection that specifies scheduled priority, any device that is unable to send packets during the scheduled portion of the network update time interval shall return this extended status code. For example, on ControlNet this code shall be returned by a node whose MAC ID is greater than maximum scheduled node (SMAX).
0x01	0x0305	<b>SCHEDULE SIGNATURE MISMATCH</b>  This extended status code shall be returned when the connection scheduling information in the originator device is not consistent with the connection scheduling information on the target network.
0x01	0x0306	<b>SCHEDULE SIGNATURE VALIDATION NOT POSSIBLE</b>  This extended status code shall be returned when the connection scheduling information in the originator device can not be validated on the target network. For example, on ControlNet this code shall be returned when there is no keeper in the master state.
0x01	0x0307 through 0x0310	Reserved by CIP
0x01	0x0311	<b>PORT NOT AVAILABLE</b>  A Port specified in a Port Segment is Not Available or does not exist.

<b>General Status</b>	<b>Extended Status</b>	<b>Explanation and Description</b>
0x01	0x0312	<p><b>LINK ADDRESS NOT VALID</b></p> <p>Link Address specified in Port Segment Not Valid</p> <p>This extended status code is the result of a port segment that specifies a link address that is not valid for the target network type. This extended status code shall not be used for link addresses that are valid for the target network type but do not respond.</p>
0x01	0x0313 through 0x0314	Reserved by CIP
0x01	0x0315	<p><b>INVALID SEGMENT IN CONNECTION PATH</b></p> <p>Invalid Segment Type or Segment Value in Connection Path</p> <p>This extended status code is the result of a device being unable to decode the connection path. This could be caused by an unrecognised path type, a segment type occurring unexpectedly, or a myriad of other problems in the connection path.</p>
0x01	0x0316	<p><b>ERROR IN FORWARD CLOSE SERVICE CONNECTION PATH</b></p> <p>The path in the Forward_Close service does not match the connection being closed. This means the connection points to a different module or application than is specified in the path. The connection is deleted but the error message shall be returned.</p>
0x01	0x0317	<p><b>SCHEDULING NOT SPECIFIED</b></p> <p>Either the Schedule Network Segment was not present or the Encoded Value in the Schedule Network Segment is invalid (0).</p>
0x01	0x0318	<p><b>LINK ADDRESS TO SELF INVALID</b></p> <p>Under some conditions (depends on the device), a link address in the Port Segment which points to the same device (loopback to yourself) is invalid.</p>
0x01	0x0319	<p><b>SECONDARY RESOURCES UNAVAILABLE</b></p> <p>In a dual chassis redundant system, a connection request that is made to the primary system shall be duplicated on the secondary system. If the secondary system is unable to duplicate the connection request, then this extended status code shall be returned.</p>
0x01	0x031A	<p><b>RACK CONNECTION ALREADY ESTABLISHED</b></p> <p>A request for a module connection has been refused because part of the corresponding data is already included in a rack connection.</p>
0x01	0x031B	<p><b>MODULE CONNECTION ALREADY ESTABLISHED</b></p> <p>A request for a rack connection has been refused because part of the corresponding data is already included in a module connection.</p>
0x01	0x031C	<p><b>MISCELLANEOUS</b></p> <p>This extended status is returned when no other extended status code applies for a connection related error.</p>

General Status	Extended Status	Explanation and Description
0x01	0x031D	<p>REDUNDANT CONNECTION MISMATCH</p> <p>This extended status code shall be returned when the following fields do not match when attempting to establish a redundant owner connection to the same target path:</p> <ul style="list-style-type: none"> <li>- O-&gt;T_RPI;</li> <li>- O-&gt;T_connection_parameters;</li> <li>- T-&gt;O_RPI;</li> <li>- T-&gt;O_connection_parameters;</li> <li>- xport_type_and_trigger.</li> </ul>
0x01	0x031E	<p>NO MORE USER CONFIGURABLE LINK CONSUMER RESOURCES AVAILABLE IN THE PRODUCING MODULE</p> <p>A target shall return this extended status when the configured number of consumers for a producing application are already in use.</p>
0x01	0x031F	<p>NO MORE USER CONFIGURABLE LINK CONSUMER RESOURCES AVAILABLE IN THE PRODUCING MODULE</p> <p>A target shall return this extended status when there are no consumers configured for a producing application to use.</p>
0x01	0x0320 – 0x07FF	Vendor specific
0x01	0x800	Network link in path to module is offline
0x01	0x801 – 0x80F	Reserved by CIP
0x01	0x810	<p>NO TARGET APPLICATION DATA AVAILABLE</p> <p>This extended status code is returned when the target application does not have valid data to produce for the requested connection. Only the target side of a connection shall return this extended status code.</p>
0x01	0x811	<p>NO ORIGINATOR APPLICATION DATA AVAILABLE</p> <p>This extended status code is returned when the originator application does not have valid data to produce for the requested connection. Only the originator side of a connection shall indicate this extended status code.</p>
0x01	0x812	<p><b>NODE ADDRESS HAS CHANGED SINCE THE NETWORK WAS SCHEDULED</b></p> <p>A router on a scheduled network (e.g.: ControlNet) has a different node address than the value configured in the connection originator.</p>
0x01	0x813	<p>NOT CONFIGURED FOR OFF-SUBNET MULTICAST</p> <p>A multicast connection has been requested between a producer and a consumer that are on different subnets, and the producer is not configured for off-subnet multicast.</p>
0x01	0x0814 through 0xFCFF	Reserved by CIP

General Status	Extended Status	Explanation and Description
0x09	Index to Element	<p>ERROR IN DATA SEGMENT.</p> <p>This general status code shall be returned when there is an error in the data segment of a forward open.</p> <p>The Extended Status shall be the index to where the error was encountered in the Data Segment (Reference section C-1.4.5 – DATA SEGMENT).</p>
0x0C	Optional	<p>OBJECT STATE ERROR</p> <p>This general status code shall be returned when the state of the target object of the connection prevents the service request from being handled. The Extended Status reports the object's present state. The extended status is optional.</p> <p>For example, a target (application) object of the connection may need to be in an edit mode before attributes can be set. This is different from a service being rejected due to the state of the device.</p>
0x10	Optional	<p>DEVICE STATE ERROR</p> <p>This general status code shall be returned when the state of the device prevents the service request from being handled. The Extended Status reports the device's present state. The extended status is optional.</p> <p>For example, a controller may have a key switch which when set to the "hard run" state causes Service Requests to several different objects to fail (i.e. program edits). This general status code would then be returned.</p>

### 3-6

### Application Connection Type using Class 0 or 1 Transports

The Forward Open service of the Connection Manager provides the ability to create two network connections at the same time, one Class 0/1 connection in the O=>T direction and the other Class 0/1 connection in the T=>O direction. When two connections are created in the same Forward Open service request certain behaviors are associated between the two connections depending on their application connection type.

The application type shall determine the target behavior concerning the relationship between different connections each sharing a producer, and shall be one of LISTEN\_ONLY, INPUT\_ONLY, EXCLUSIVE\_OWNER, or REDUNDANT\_OWNER.

A further element in applying these connections is the real time formats, including Run/Idle notification.

### **3-6.1 Real time formats including RUN/IDLE notification**

Each class 0/1 network connection shall send packets using one of the following real time formats:

- Modeless format – no run/idle notification.
- Zero length data format indicates idle.
- Heartbeat format – no run/idle notification.
- 32-bit header format includes run/idle notification.

The real time format supported by a target device is indicated in the Connection Manager section, ConnectionN entry, Connection Parameters field of its EDS file. This allows a connection originator configuration tool to know the O=>T and T=>O real time formats for each connection.

The connection originator needs to know the real time formats to produce data using the correct format and to correctly decode the data it is consuming. The O=>T and T=>O real time formats are specified in instance attribute 2 of the Connection Configuration Object.

The target device needs to know the real time format to decode the data it is consuming and to produce data using the correct format. The target device knows the real time formats based on the Connection\_Path specified in the Forward\_Open, each unique Connection\_Path is mapped to a O=>T real time format and a T=>O real time format.

#### **3-6.1.1 Modeless Format**

The modeless real time format may include 0-n bytes of application data and there is no run/idle notification included with this real time format. The modeless real time format network connection may be either fixed or variable size.

A class 0 modeless real time packet format is:

0-n bytes of application data
-------------------------------

A class 1 modeless real time packet format is:

2 byte sequence count	0-n bytes of application data
-----------------------	-------------------------------

#### **3-6.1.2 Zero Length Data Format**

When the zero length data real time format includes 1-n bytes of application data the producer is indicating run mode. When the zero length data real time format includes 0 bytes of application data the producer is indicating idle mode. The zero length data format network connection shall be variable size.

A class 0 zero length data real time packet format showing idle mode is:

0 bytes of application data
-----------------------------

A class 0 zero length data real time packet format showing run mode is:

1-n bytes of application data
-------------------------------

A class 1 zero length data real time packet format showing idle mode is:

2 byte sequence count
-----------------------

A class 1 zero length data real time packet format showing run mode is:

2 byte sequence count	1-n bytes of application data
-----------------------	-------------------------------

### 3-6.1.3 Heartbeat Format

The heartbeat real time format includes 0 bytes of application data and there is no run/idle notification included with this real time format. The modeless real time format network connection shall be fixed size.

A class 0 heartbeat real time packet format is:

0 bytes of application data
-----------------------------

A class 1 heartbeat real time packet format is:

2 byte sequence count	0 bytes of application data
-----------------------	-----------------------------

### 3-6.1.4 32-Bit Header Format

The 32 bit header real time format includes 0-n bytes of application data prefixed with 32 bits of header. The modeless real time format network connection may be either fixed or variable size.

The 32-bit real time header format prefixed to the real-time data shall be the following form:

**Table 3-6.1 32-Bit Real Time Header**

Bits 4-31	Bits 2-3	Bit 1	Bit 0
Reserved	ROO	COO	Run/Idle

The run/idle flag (bit 0) shall be set (1 = RUN) to indicate that the following data shall be sent to the target application. It shall be clear (0 = IDLE) to indicate that the idle event shall be sent to the target application. The ROO and COO fields (bits 1-3) are defined in 3-6.5.4, Redundant Owner. The reserved field (bits 4-31) shall be reserved and set to 0.

A class 0 32 bit header real time packet format is:

32-bit real time header	0-n bytes of application data
-------------------------	-------------------------------

A class 1 zero length data real time packet format is:

2 byte sequence count	32-bit real time header	0-n bytes of application data
-----------------------	-------------------------	-------------------------------

## 3-6.2 Configuration

Any application type connection may send configuration with a Forward\_Open. If present the configuration shall be specified in a data segment and shall be delivered to the component specified in the configuration path. If no data segment is specified the existing configuration continues to be used. If a data segment is specified and no other active I/O connection which specified this component exists, the configuration in the data segment shall be applied. If a data segment is specified and another I/O connection to this component exists the configuration in the data segment shall be compared to the existing configuration; if there are no differences the connection may be opened, if there are differences an Ownership Conflict error (0x01,0x106) shall be returned.

### **3-6.3 Specifying Different Application Connection Types**

The EDS section (chapter 7) of this specification defines how to declare the application connection type for a connection. Software tools can use this information to audit and enforce connection type issues if desired (issues like multiple exclusive owner connections to the same target, only listen only connections to a target, exclusive owner connection with redundant owner connection to the same target, etc.).

There is no requirement for connection originator devices or bridge/router devices to take action based on connection type (other than redundant owner connections), so the connection type is not explicitly included in the forward open data (with the exception of redundant owner connections). A specific connection path shall have a single application connection type. The target device can then take the appropriate actions for the connection based on the application connection type definitions below.

### **3-6.4 Application types**

The requirements for each of the 4 connection types, Listen Only, Input Only, Exclusive Owner and Redundant Owner are detailed in this section.

#### **3-6.4.1 Listen only**

If a connection has an application type of listen only, it shall be dependent on a non-Listen only application connection for its existence. For a scheduled listen only connection, the Forward\_Open path shall contain a schedule segment. The O=>T connection shall use the heartbeat format as described in section 3-6.1.3. A target may accept multiple listen only connections which specify the same T=>O path. Devices that wish to listen to multicast data without providing configuration may use this application type. If the last connection on which a listen only connection depends is closed or times out, the target device shall stop sending the T=>O data which will result in the listen only connection being timed out by the originator device.

#### **3-6.4.2 Input only**

If a connection has an application type of input only, it shall not be dependent on any other connection for its existence. For a scheduled input only connection, the Forward\_Open path shall contain a schedule segment. The O=>T connection shall use the heartbeat format as described in section 3-6.1.3. A target may accept multiple input only connections which specify the same T=>O path. A specific implementation may limit the number of input only connections it accepts. In addition, the target may accept listen only connections that use the same multicast T=>O data.

#### **3-6.4.3 Exclusive Owner**

If a connection has an application type of exclusive owner, it shall not be dependent on any other connection for its existence. For scheduled exclusive owner connections, the Forward\_Open path shall contain a schedule segment. O=>T application data that controls outputs may be present. A target may only accept one exclusive owner connection which specifies the same O=>T path. In addition, the target may accept listen only and input only connections that use the same multicast T=>O data.

The term connection owner shall refer to the connection originator whose O=>T packets are being consumed by the target object. The term owning connection shall refer to the connection associated with connection owner.

When an exclusive owner connection timeout occurs in a target device, the target device shall stop sending the associated T=>O data. The T=>O data must not be sent even if one or more input only connections exist. This requirement exists to signal the originator of the exclusive owner connection that the O=>T data is no longer being received by the target device.

NOTE: One possible way to prevent an exclusive owner connection timeout in a target device from stopping the T=>O production is for the target device to also support production of the T=>O data as point to point for the exclusive owner connection.

#### **3-6.4.4 Redundant owner**

##### **3-6.4.4.1 General**

The redundant owner connection shall allow multiple separate originator applications to each establish an independent, identical connection to the transport of a target application. The target transport shall in turn send events to the target application so that the redundant owner connection appears as a single, exclusive owner connection to the target application. At most one of the originator applications shall have its data applied to the target application. The target transport shall multicast the target application data to each of the originator applications.

When a single redundant owner connection exists and a redundant owner connection timeout occurs in a target device, the target device shall stop sending the associated T=>O data. The T=>O data must not be sent even if one or more input only connections exist. This requirement exists to signal the originator of the redundant owner connection that the O=>T data is no longer being received by the target device.

NOTE: One possible way to prevent a redundant owner connection timeout in a target device from stopping the T=>O production is for the target device to also support production of the T=>O data as point to point for the redundant owner connection.

NOTE: The redundant owner connection may commonly be used in applications where up time is at a premium. In these applications, multiple originator applications can each establish a connection to a target application (typically an output application). Should an originator application fail or otherwise give up control, another originator application can quickly have its data applied to the target application without having to establish a connection with the target.

#### **3-6.4.4.2 Establishing the Connection**

A redundant owner connection and an exclusive owner connection shall not both be established to a target application at the same time. Multiple redundant owner connections may be established to the same target simultaneously provided that the following fields of the Forward\_Open request are identical and the connection paths match. The connection path shall match including any data segments.

- O2T\_RPI;
- O2T\_connection\_parameters;
- T2O\_RPI;
- T2O\_connection\_parameters;
- xport\_type\_and\_trigger.

The target shall return general status = 0x01 and extended status = 0x031D if any of these fields do not match.

The target transport shall only send the CM\_open\_indication to the target application when the first Forward\_Open is received. Subsequent redundant Forward\_Opens shall not cause a CM\_open\_indication to be sent to the target application.

NOTE: Bit 15 of the Forward Open request.O->T\_connection\_parameters specifies whether the connection is an exclusive owner or redundant owner connection

#### **3-6.4.4.3 Redundant owner O->T data format**

##### **3-6.4.4.3.1 General**

The Redundant owner shall use the 32-bit header format described in section 3-6.1.4.

##### **3-6.4.4.3.2 Claim Output Ownership (COO) Flag**

The COO flag shall be set (1) when an originator application wants its connection to be the owning connection of the target application. The COO flag shall be reset (0) when an originator application does not want its connection to be the owning connection of the target application. When the owning connection resets (0) its COO flag, its sibling connections shall be checked for a set (1) COO flag. The new owner shall be any of the connections that have their COO flag set.

NOTE: This results in undefined behavior if more than one other connection has its COO flag set.

##### **3-6.4.4.3.3 Ready for Ownership of Outputs (ROO) Priority Value**

The ROO priority value shall be non-zero when an originator application does not want to force its connection to be the owning connection of the target application, but is ready to be the owning connection should there be no originator applications claiming to be the owning connection. The ROO priority value shall be zero when the originator application does not want to be the owning connection of the target connection and is not to be the owning connection should there be no originator application claiming to be the owning connection. The ROO priority value shall be used only when the COO flag is reset.

The value of the ROO field can range from 0 to 3. The originator applications shall each determine a unique non-zero ROO value.

#### **3-6.4.4.4 Determining the owning connection**

The originator applications shall determine among themselves which originator application has the owning connection. The owning connection shall be determined by the originator application that sets its COO flag. In situations where multiple originator applications have their COO flag set or where no originator connections have their COO flag set, the following rules shall be applied by the target transport to determine the owning connection.

- There shall be no owning connection until an originator application sends a real-time packet with the COO flag set;
- If there is only one originator application which had the COO flag set in its last realtime packet, that originator application shall have the owning connection;
- If there are multiple originator applications which had the COO flag set in its last realtime packet, the last originator application that transitioned its COO flag from reset to set shall have the owning connection.
- If the originator application with the owning connection resets its COO flag, closes its connection, or if that connection times out, and no other originator applications have their COO flags set, the originator application with the highest non-zero ROO priority value shall have the owning connection.
- If all of the originator applications have their COO flags reset and ROO priority values set to zero, there shall be no owning connection.
- When the first real-time packet containing a set COO flag is received by the target transport, the originator application that sent the real-time packet shall have the owning connection.

#### **3-6.4.4.5 Transporting events and data to a target application**

The connection related events from each of the redundant owner connections shall be combined using the following rules such that the target application sees only a single exclusive owner connection:

- Until an owning connection is initially determined, the transport shall not indicate realtime data reception to the target application;
- If an owning connection is determined, the transport shall indicate the event consistent with the owning connections real-time data to the target application. If the Run/Idle flag is reset in the real-time data for the owning connection, the transport shall indicate the idle state to the target application. If the Run/Idle flag is set in the real-time data for the owning connection, the transport shall indicate the run state and the real-time data to the target application;
- If an owning connection had been previously determined, but no originator is currently claiming or ready for ownership, the transport shall indicate idle state to the target application;
- If all the redundant connections are closed or have been timed out, the target transport shall indicate the event consistent with the last connection to have been closed or timed out to the target application.

## 3-7 Port Object Class Definition

### Class Code: F4 hex

The Port Object enumerates the CIP ports present on the device. One instance exists for each CIP port.

#### 3-7.1 Port Object Class Attributes

The Port Object Class attributes are defined below in Table 3-7.1.

**Table 3-7.1 Port Class Attributes**

Attr ID	Need In Implementation	Access Rule	Attribute Name	Data Type	Attribute Description	Semantics of Values
1	This class attribute is optional and is described in Chapter 4 of this specification.					
2	Required	Get	Max Instance	UINT	Maximum instance number.	
3	Required	Get	Num Instances	UINT	Number of ports currently instantiated.	
4 – 7	These class attributes are optional and are described in Chapter 4 of this specification.					
8	Required	Get	Entry Port	UINT	Returns the instance of the Port Object that describes the port through which this request entered the device.	
9	Required	Get	Port Instance Info	ARRAY of STRUCT of	Array of structures containing instance attributes 1 and 2 from each instance.	The array is indexed by instance number, up to the maximum number of instances. The values at index 1 (offset 0) and any non-instantiated instances shall be zero.
			Port Type	UINT	Enumerates the type of port	See instance attribute #1
			Port Number	UINT	CIP port number associated with this port	See instance attribute #2

### 3-7.2 Port Object Class Services

The Port Class supports the following CIP Common Services:

**Table 3-7.2 Port Class Services**

Service Code	Need In Implementation	Service Name	Service Description
01hex	Optional	Get_Attributes_All	Returns the contents of all attributes of the class.
0Ehex	Conditional	Get_Attribute_Single	Used to read a Port Class attribute value. This service is Required if any of the Port Class Attributes are supported.

### 3-7.3 Port Object Instance Attributes

The following list provides a summary of the Port Instance attributes and their associated data types.

**Table 3-7.3 Port Object Instance Attributes**

Attr ID	Need In Implem	Access Rule	NV	Attribute Name	Data Type	Description of Attribute	Semantics
1	Required	Get	NV	Port Type	UINT	Enumerates the type of port	0 = connection terminates in this device 1 = reserved for compatibility with existing protocols 2 = ControlNet 3 = ControlNet redundant 4 = EtherNet/IP <sup>3</sup> 5 = DeviceNet 6 – 99 = reserved for compatibility with existing protocols 100 – 199 = Vendor Specific 200 = CompoNet 201 = Modbus/TCP 202 = Modbus/SL 203 – 65534 = Reserved for future use 65535 = unconfigured port
2	Required	Get	NV	Port Number	UINT	CIP port number associated with this port	Manufacturer assigns a unique value to identify each communication port. Value 1 is reserved for internal product use (ie. backplane).

Attr ID	Need In Implem	Access Rule	NV	Attribute Name	Data Type	Description of Attribute	Semantics
3	Required	Get	NV	Link Object	STRUCT of		
				Path Length	UINT	Number of 16 bit words in the following path	Range = 2 - 6
				Link Path	Padded EPATH	Logical path segments that identify the object for this port. For example, this could be the TCP/IP Interface Object.	The path shall consist of one logical class segment and one logical instance segment. The maximum size is 12 bytes. See Appendix C, Logical Segments.
4	Required	Get	NV	Port Name	SHORT_STRING	String which names the physical network port. The maximum number of characters in the string is 64. For example, this may be "Port A".	This value is unique for each physical network port. If multiple CIP ports use the same physical network port, they shall each have the same Port Name value. Likewise, a CIP port shall not have the same Port Name value as any other CIP port if they do not share the same physical network port.
5	Optional	Get	NV	Port Type Name	SHORT_STRING	String which names the port type. The maximum number of characters in the string is 64. For example, this may be "DeviceNet".	
6	Optional	Set	NV	Port Description	SHORT_STRING	String which describes the port. The maximum number of characters in the string is 64. For example, this may be "Product Line 22".	
7	Required	Get	NV	Node Address <sup>1</sup>	Padded EPATH	Node number of this device on port. The range within this data type is restricted to a Port Segment.	The encoded port number shall match the value presented in attribute 2.
8	Conditional	Get	NV	Port Node Range <sup>2</sup>	STRUCT of		
				Minimum Node Number	UINT	Minimum node number on port.	
				Maximum Node Number	UINT	Maximum node number on port.	
9	Optional	Get	NV	Port Key	Packed EPATH	Electronic key of network/chassis this port is attached to. This attribute shall be limited to format 4 of the Logical Electronic Key segment.	

Attr ID	Need In Implem	Access Rule	NV	Attribute Name	Data Type	Description of Attribute	Semantics
1				A device which does not have a node number on the port can indicate a zero length node address within the Port Segment (0x10 0x00).			
2				If a device can report its port characteristics within the range allowed (e.g. DeviceNet MACID) then it shall support this attribute. Otherwise (e.g. EtherNet/IP IP Address) it shall not support this attribute.			
3				<b>This port type was formerly known as TCP/IP</b>			

### 3-7.4 Port Object Instance Common Services

The Port Object Instance supports the following CIP Common services:

**Table 3-7.4 Port Object Instance Common Services**

Service Code	Need In Implementation	Service Name	Service Description
01hex	Optional	Get_Attributes_All	Returns the contents of all attributes of the class.
05hex	Optional	Reset	
0Ehex	Conditional	Get_Attribute_Single	Used to read a Port Object instance attribute. This service is Required if any of the Port Instance Attributes are supported.
10hex	Optional	Set_Attribute_Single	

#### 3-7.4.1 Get\_Attributes\_All Response

The Get\_Attributes\_All response for the class attributes shall concatenate attributes 1, 2, 3, 8 and 9 in that order. If class attribute 1 (Revision) is not supported, then a default value of one (1) shall be returned. The Get\_Attribute\_All response for the instance attributes shall concatenate attributes 1, 2, 3,4 and 7 in that order.

This page is intentionally left blank

## **Volume 1: Common Industrial Protocol (CIP<sup>TM</sup>) Specification**

### **Chapter 4: CIP Object Model**

## Contents

4-1	Introduction.....	3
4-2	Description.....	4
4-3	Class Code .....	4
4-4	Attributes .....	4
4-4.1	Class Attributes.....	5
4-4.2	Instance Attributes .....	7
4-5	Common Services .....	7
4-5.1	Get_Attributes_All Response .....	9
4-5.2	Set_Attributes_All Request.....	9
4-5.3	Optional Services .....	10
4-6	Object-specific Services.....	10
4-6.1	Service Parameters.....	12
4-6.2	Service Response Data.....	12
4-7	Behavior.....	12
4-7.1	State .....	13
4-7.2	Event .....	14
4-8	Accessing Application object data.....	14
4-8.1	Access Through Explicit Messaging Connections.....	15
4-8.2	Access Through I/O Connections .....	15
4-9	Extending Existing Objects and Defining New Objects.....	17
4-9.1	CIP Object Address Ranges.....	18
4-9.2	Making Extensions to Objects .....	19
4-9.2.1	Vendor-specific Extensions.....	19
4-9.2.1.1	Example .....	20
4-9.2.1.2	Implementing a Vendor-specific Extension .....	21
4-9.2.2	Open Extensions .....	22
4-9.3	Defining a New Object .....	23
4-9.3.1	Vendor-specific .....	23
4-9.3.2	Open .....	24
4-9.4	New Common Service .....	25

## **4-1 Introduction**

This chapter includes an explanation of how to read the object specifications in the library. Each object is defined using the same format.

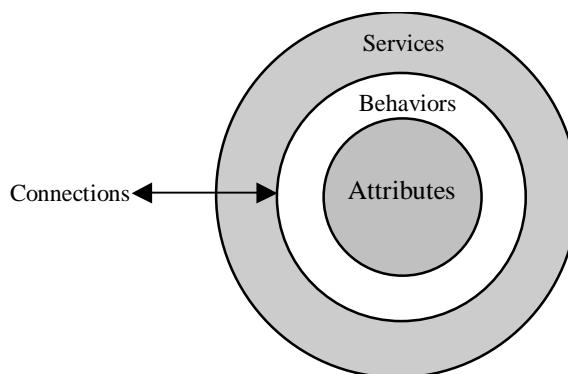
<b>For information about</b>	<b>Go to Section</b>
Reading an Object Specification	4-1
Description of Object	4-2
Class Code	4-3
Attributes	4-4
Common Services	4-5
Object-specific Services	4-6
Behavior	4-7
Connections	4-8

Each specification contained in the library is defined based on the contents of an object.

An object consists of the following. See Figure 4-1.1.

- a set of closely related attributes (data)
- a set of behaviors
- a set of services (common or object-specific)
- a set of connections

**Figure 4-1.1 Defining an Object Specification**



The objects in the CIP Application Object Library are defined in the following terms:

**Table 4-1.1 Object Definitions**

<b>Description</b>	a description of the object being specified
<b>Class Code</b>	a hexadecimal identifier assigned to each CIP object
<b>Attributes</b>	the data associated with this object
<b>Common Services</b>	a list of the common services defined for this object
<b>Object-specific Services</b>	the full specifications of any services unique to this object
<b>Connections</b>	connections supported by this object
<b>Behavior</b>	the relationship between attribute values and services

## **4-2 Description**

Every object specification begins with a brief functional definition of the object being defined. For example, the Description of the Identity Object reads: *This object provides identification of and general information about the device.*

## **4-3 Class Code**

This part of the definition is a hexadecimal value unique to an object. Use the class code to identify the object class when accessing objects in devices. Open DeviceNet Vendor Association and ControlNet International are responsible for allocation and coordination of the class codes. However, managing Vendor Specific class codes and guaranteeing the values are unique to a Vendor ID is the responsibility of individual member companies.

In addition to the individual object definition, a summary list of all the objects and their class codes is located at the beginning of the Object Library.

## **4-4 Attributes**

The Attribute part of an object specification is divided into two sections:

- Class attributes
- Instance attributes

#### 4-4.1 Class Attributes

A *Class Attribute* is an attribute that is shared by all objects within the same class. Class Attributes are defined using the following terms:

**Table 4-4.1 Class Attribute Fields**

Attribute ID	Need in Implementation	Access Rule	NV	Name	CIP Data Type	Description of Attribute	Semantics of Values
1	2	3	4	5	6	7	8

1. **Attribute ID** is an integer identification value assigned to an attribute. Use the Attribute ID in the Get\_Attributes and Set\_Attributes services list. The Attribute ID identifies the particular attribute being accessed.
2. **Need in Implementation** specifies whether or not the attribute is necessary in the object class implementation. An attribute may be *Optional*, *Required* or *Conditional*.

A conditional attribute is required if certain object behaviors and/or attributes are implemented as defined by the class or within the Device Profile.

**Important:** If a Class Attribute is *optional*, then you must define a default value.

If the default value of an optional attribute does not match the behavior of the object when the object does not implement the attribute, the object definition shall declare the behavior.

In all cases, the term “default” indicates a “factory default” as shipped from the vendor.

3. The **Access Rule** specifies how a requestor can access an attribute. The definitions for access rules are:
  - Settable (Set) – The attribute can be accessed by one of the *Set\_Attribute* services. If the behavior of your device does not require a *Set\_Attribute* service, then you are not required to implement the attribute as settable.  
**Important:** Settable attributes can also be accessed by *Get\_Attribute* services.
  - Gettable (Get) – The attribute can be accessed by one of the *Get\_Attribute* services.
4. **NV** indicates whether an attribute value is maintained through power cycles. This column is used in object definitions where non-volatile storage of attribute values is required. An entry of ‘NV’ indicates value shall be saved, ‘V’ means not saved.
5. **Name** refers to the attribute.
6. **Data Type** is used in the *Get\_Attribute* and *Set\_Attribute* services. You must follow the specified data type for all products using the attribute being defined. CIP data types and their ranges are defined in Appendix C.
7. **Description of Attribute** provides general information about the attribute.
8. **Semantics of Values** specifies the meaning of the value of the attribute.

**Important:** Seven Class Attribute IDs are reserved for class object definitions. They are:

- Revision
- Max Instance
- Number of Instances
- Optional Attribute list
- Optional Service list
- Maximum Number Class Attributes
- Maximum Number Instance Attributes

Because these attributes are reserved, attribute ID numbers 1 through 7 are **always** reserved. Therefore, if you want to add a class attribute to an object definition, you must start with attribute ID #8. Unless otherwise specified in an object definition, each class attribute numbered 1 through 7 exists as an optional attribute within each class.

The seven reserved Class Attributes have the following definitions:

**Table 4-4.2 Reserved Class Attributes for All Object Class Definitions**

Attribute ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1	Conditional *	Get	Revision	UINT	Revision of this object	The current value assigned to this attribute is one (01). If updates that require an increase in this value are made, then the value of this attribute increases by 1.
2	Optional	Get	Max Instance	UINT	Maximum instance number of an object currently created in this class level of the device.	The largest instance number of a created object at this class hierarchy level.
3	Optional	Get	Number of Instances	UINT	Number of object instances currently created at this class level of the device.	The number of object instances at this class hierarchy level.
4	Optional	Get	Optional attribute list	STRUCT of	List of optional instance attributes utilized in an object class implementation.	A list of attribute numbers specifying the optional attributes implemented in the device for this class.
			number of attributes	UINT	Number of attributes in the optional attribute list.	The number of attribute numbers in the list.
			optional attributes	ARRAY of UINT	List of optional attribute numbers.	The optional attribute numbers.

Attribute ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
5	Optional	Get	Optional service list	STRUCT of	List of optional services utilized in an object class implementation.	A list of service codes specifying the optional services implemented in the device for this class.
			number services	UINT	Number of services in the optional service list.	The number of service codes in the list.
			optional services	ARRAY of UINT	List of optional service codes.	The optional service codes.
6	Optional	Get	Maximum ID Number Class Attributes	UINT	The attribute ID number of the last class attribute of the class definition implemented in the device.	
7	Optional	Get	Maximum ID Number Instance Attributes	UINT	The attribute ID number of the last instance attribute of the class definition implemented in the device.	

\* If the value is 01, then this attribute is OPTIONAL in implementation. If the value is greater than 01, then this attribute is REQUIRED.

#### 4-4.2 Instance Attributes

An *Instance Attribute* is an attribute that is unique to an object instance and not shared by the object class.

Instance Attributes in the Object Library are defined in the same terms as Class Attributes. There are no reserved Instance Attributes.

**Table 4-4.3 Instance Attribute Fields**

Attribute ID	Need in Implementation	Access Rule	NV	Name	CIP Data Type	Description of Attribute	Semantics of Values
1	2	3	4	5	6	7	8

#### 4-5 Common Services

Common Services are those whose request/response parameters and required behaviors are defined in CIP Common, Appendix A.

The Common Service component of an object definition includes:

**Table 4-5.1 Common Service Fields**

Service	Need in Implementation		Service	Description of	
	Code	Class	Instance	Name	Service
1	2	3	4	5	

1. **Service Code** is the hexadecimal value assigned to each CIP service.
- 2-3. **Need in implementation** specifies whether or not the service is needed in the implementation of this object at the **Class** level or at the **Instance** level. In these columns will appear one of four specifications:

- Optional; or
- Required; or
- Conditional; or
- Not applicable (“n/a”)

A conditional service is required if certain object behaviors, attributes and/or services are implemented as defined by the object. All conditional services shall specify the condition(s) that qualify the service to be required. If the need in implementation is specified as Not Applicable at the class or instance level, it shall not be supported at that level.

**Important:** If an optional service is implemented in a class, and the optional Service List class attribute is also implemented in the class, the service shall be included in the Service List.

Services trigger the Behavior of an object based on the values of the attributes accessed by the service. Common Services can be directed to either the Class level or the Instance level of an object, which may produce different behavior at each level.

- *Class Level*: behavior triggered by services sent to the Object Class.
- *Instance Level*: behavior triggered by services sent to the Object *Instance*.

**Table 4-5.2 Common Service Naming Conventions**

Common Services sent to	Are called
the Class Level of an object	Common Class Level Services
the Instance Level of an object	Common Instance Level Services

4. **Service Name** refers to the service. See CIP Common, Appendix A for a complete list of CIP common services.
5. **Description of Service** provides a brief definition of the service.

#### 4-5.1 Get\_Attributes\_All Response

When the Get\_Attributes\_All common service is included in the list of supported common services, then the Get\_Attributes\_All response **shall** be included in the object definition. This specifies the **structure** of the data returned in the Service Data portion of Explicit Response Message. (This is the Response\_Data field described in the Message Router Response Format. See Chapter 2 for the definition.) The following byte array is an example of how the **structure** of the Service Data portion of a Get\_Attributes\_All response is typically specified:

**Table 4-5.3 Example Get\_Attributes\_All Response Data – Class Level**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0								Revision (low byte) Default = 1
1								Revision (high byte) Default = 0
2								Max Instance (low byte) Default = 0
3								Max Instance (high byte) Default = 0
4								Number of Instances (low byte) Default = 0
5								Number of Instances (high byte) Default = 0
6								Optional Attribute List : number of attributes (low byte) Default = 0
7								Optional Attribute List : number of attributes (high byte) Default = 0
8								Optional Attribute List : optional attribute #1 (low byte)
9								Optional Attribute List : optional attribute #1 (high byte)
n								Optional Attribute List : optional attribute #m (low byte)
n+1								Optional Attribute List : optional attribute #m (high byte)

**Important:** Insert default values for all unsupported attributes.

The following rules are to be adhered to when specifying an object's Get\_Attributes\_All response format:

- Default values **shall** be supplied for all "optional" attributes. If a product chooses not to implement some of the optional attributes it will be required to insert the specified default value for the unimplemented attribute.
- If new attributes are added to **the response structure**, those attributes **shall** be added to the end.

For CIP defined (public) objects, the definition of the Get\_Attributes\_All response data is controlled by CIP. Inclusion of vendor specific or any other data in the response data is not allowed.

#### 4-5.2 Set\_Attributes\_All Request

When the Set\_Attributes\_All common service is included in the list of supported common services, then the Set\_Attributes\_All request must be included in the object definition. This specifies the sequence or order of the data supplied in the Service Data portion of the request. (This is the Request\_Data field described in the Message Router Request Format. See Chapter 2 for the definition.) The following byte array is an example of how the format of the Service Data portion of a Set\_Attributes\_All request is typically specified:

At the **Instance level**, the order of attributes passed in the Set\_Attributes\_All request is as follows:

**Table 4-5.4 Set\_Attributes\_All Request Data**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0								Output Range
1								Value Data Type
2								Fault State
3								Idle State
4								Fault Value (low byte)
5								Fault Value (high byte)
6								Idle Value (low byte)
7								Idle Value (high byte)

For CIP defined objects, the definition of the Set\_Attributes\_All service data is controlled by CIP. Inclusion of vendor specific or any other data in the service data is not allowed.

### **4-5.3 Optional Services**

All objects shall have the following services defined as Optional at both the Class and Instance level unless otherwise specified by an object class as Conditional or Required. Services not listed here or in the object definition shall not be supported.

- Get\_Attribute\_Single
- Set\_Attribute\_Single
- Get\_Attribute\_List
- Set\_Attribute\_List
- Multiple\_Service\_Packet
- No\_Operation (NOP)
- Get\_Member
- Set\_Member
- Insert\_Member
- Remove\_Member

### **4-6 Object-specific Services**

The section titled “Object-specific Services” provides a list of the unique services supported by this class of objects and their service codes.

Whereas Common Services can be used in many objects, Object-specific Services are unique to each object.

For example, many objects support the Get\_Attributes\_All service; however, only the *DeviceNet Object* supports Allocate\_Master/Slave\_Connection\_Set (see DeviceNet Specification, Volume 3).

The Object-specific Services component of the object class definition includes:

**Table 4-6.1 Object-specific Service Data Fields**

Service Code	Need in Implementation		Service Name	Description of Service
	Class	Instance		
1	2	3	4	5

1. **Service Code** is the hexadecimal value assigned to each CIP service.
- 2-3. **Need in implementation** specifies whether or not the service is needed in the implementation of this object at the **Class** level or at the **Instance** level. In these columns will appear one of four specifications:
  - Optional; or
  - Required; or
  - Conditional; or
  - Not applicable (“n/a”)

A conditional service is required if certain object behaviors, attributes and/or services are implemented as defined by the object. All conditional services shall specify the condition(s) that qualify the service to be required. If the need in implementation is specified as Not applicable at the class or instance level, it shall not be supported at that level.

**Important:** If a service is specified to be optional and is implemented in a device, its service code must be included in the “optional service list” class attribute.

Services trigger the Behavior of an object based on the values of the attributes accessed by the service. Object-specific Services can be directed to either the Class level or the Instance level of an object, which may produce different behavior at each level.

- Class Level: behavior triggered by services sent to the Object Class.
- Instance Level: behavior triggered by services sent to the Object *Instance*.

**Table 4-6.2 Object-specific Service Naming Convention**

Object-specific Services sent to	Are called
the Class Level of an object	Object-specific Class Level Services
the Instance Level of an object	Object-specific Instance Level Services

4. **Service Name** refers to the service.
5. **Description of Service** provides a brief definition of the service.

#### **4-6.1 Service Parameters**

Whereas each Common Service has fixed parameters, each Object-specific Service has unique parameters. This section defines those parameters for each Object-specific Service.

**Table 4-6.3 Object-specific Service Parameter Field Descriptions**

Name	Type	Description of Request Parameters	Semantics of Values
1	2	3	4

1. **Name** refers to the service request parameter.
2. **Type** specifies the data type of the service request parameter.
3. **Description of Request Parameters** describes the purpose of the request parameter.
4. **Semantics of Values** specifies the meaning of the values of the service request parameter, such as “the value is counts of microseconds.”

#### **4-6.2 Service Response Data**

Table 4-6.4 is the description of the service response data that includes the following:

**Table 4-6.4 Object-specific Service Response Data**

Name	Type	Description of Response Data	Semantics of Values
1	2	3	4

- 1) **Name** refers to the service response data.
- 2) **Type** specifies the data type of the service response data.
- 3) **Description of Response Data** describes the purpose of the response data.
- 4) **Semantics of Values** specifies the meaning of the values of the service response data.

This object definition component also includes a description of the usage and purpose of the service. If the service triggers a complex behavior, then you must specify it. Response Data from CIP Common services shall conform to that specified in CIP Common, Appendix A unless specified otherwise in the Object Definition or Device Profile.

#### **4-7 Behavior**

Behavior of an object may be triggered by an object’s services and is based on the values of the attributes accessed by the service. Together, the services and attribute values initiate state changes in the object. The behavior definition determines *how* an object responds when it receives notification of an event that changes its state. Behavior must be defined in terms of:

- the *state* an object is in when it receives notification of a state-changing event; and
- the *event* the object receives.

## 4-7.1 State

A **state** is an object's current active mode of operation (e.,g., Running, Idle).

To define behavior in these terms, use a State Event Matrix and a State Transition Diagram when applicable. A **State Event Matrix** is a table that lists all possible events and services that initiate a state change, and indicates an object's response to the event or service based on the state of the object when it receives notification of that event.

For example, the table below shows three states (other objects may or may not have other states):

- **Non-existent:** the object has not yet been created. You must implement the create service to transition the object to existent.
- **Idle:** the object accepts services (e.g., Start, Stop, Get\_Attribute\_Single), but does not produce.
- **Running:** the object is performing its function.

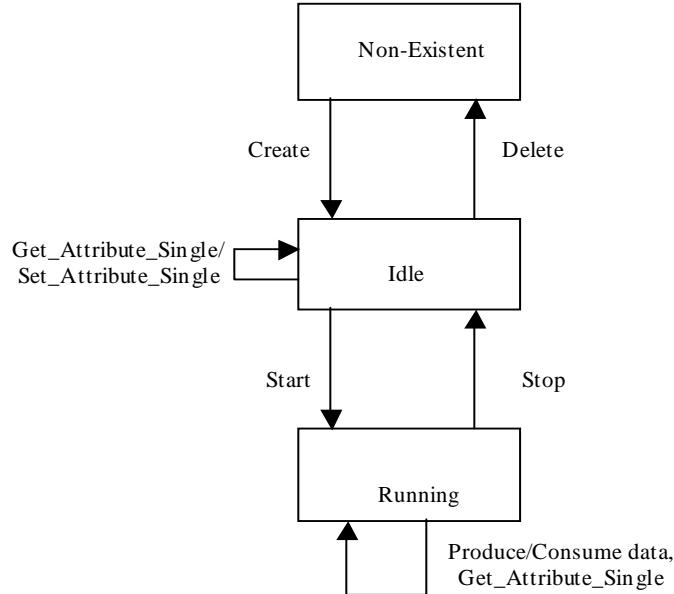
**Table 4-7.1 State Event Matrix Table Format**

Event	State		
	Non-existent	Idle	Running
Create	Transition to Idle	Error: Object already exists.	Error: Object already exists.
Delete	Error: Object does not exist. (General Error Code 0x16)	Transition to Non-Existent	Error: Object State Conflict (General Error Code 0x0C)
Start	Error: Object does not exist. (General Error Code 0x16)	Transition to Running	Error: Object State Conflict (General Error Code 0x0C)
Stop	Error: Object does not exist. (General Error Code 0x16)	Error: Object State Conflict (General Error Code 0x0C)	Transition to Idle
Get_Attribute_Single	Error: Object does not exist. (General Error Code 0x16)	Validate/service the request. Return response	Validate/service the request. Return response
Set_Attribute_Single	Error: Object does not exist. (General Error Code 0x16)	Validate/service the request. Return response	Error: Object State Conflict (General Error Code 0x0C)

## 4-7.2 Event

An **event** is an external stimulus that could cause a state transition. A **State Transition Diagram** graphically illustrates the state of an object and includes services and attributes that cause it to transition to another state.

**Figure 4-7.1 State Transition Diagram Example**



When applicable, some object behavior definitions include an **Attribute Access Table**, which lists all object attributes and how to access them in a given state.

**Table 4-7.2 Attribute Access Table Example**

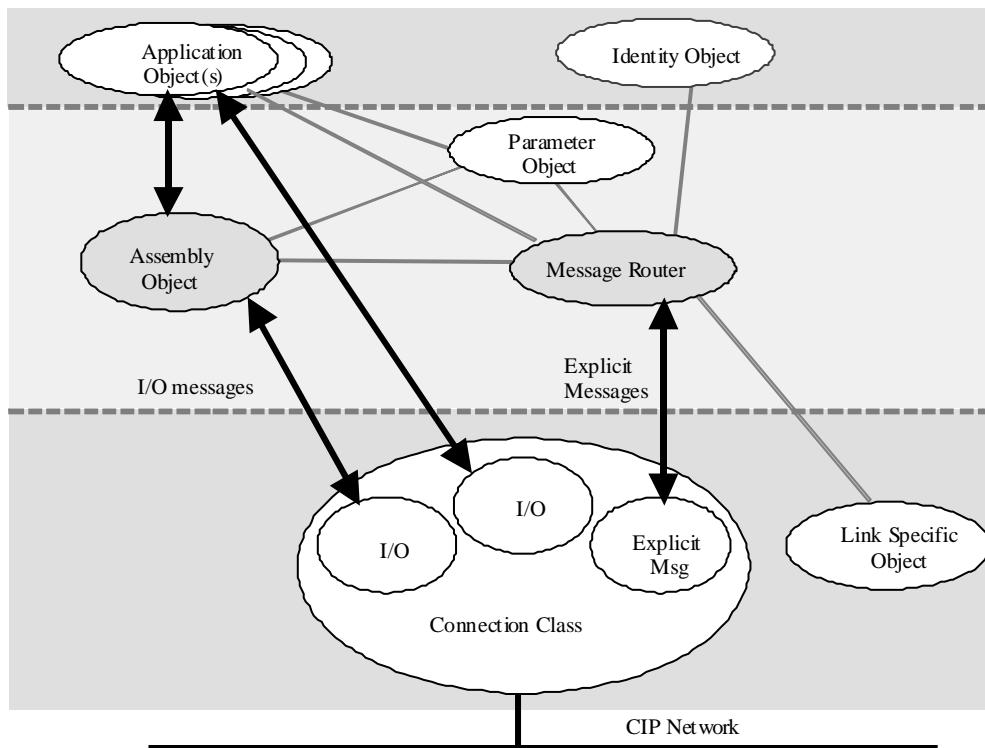
Attribute	State		
	Non-existent	Idle	Running
Number_of_Members	Not available	Read Only	Read Only
Member_List	Not available	Read Only	Read Only
Data	Not available	Read/Write	Read Only
Owner	Not available	Read Only	Read Only

## 4-8 Accessing Application object data

An Object Class definition must indicate any special provisions that have been made with regards to the external access of its associated data.

As described in Chapter 1 of Volume 1, two types of Connections exist; Implicit (I/O) and Explicit Messaging. This section provides an overview of how these Connection types relate to Application Objects and how they can be used to access Application Object data. Figure 4-8.1 presents an overview of the relationship between the Connection Class and Application Objects and serves as a prefix to the overview.

**Figure 4-8.1 Connection Paths**



#### 4-8.1 Access Through Explicit Messaging Connections

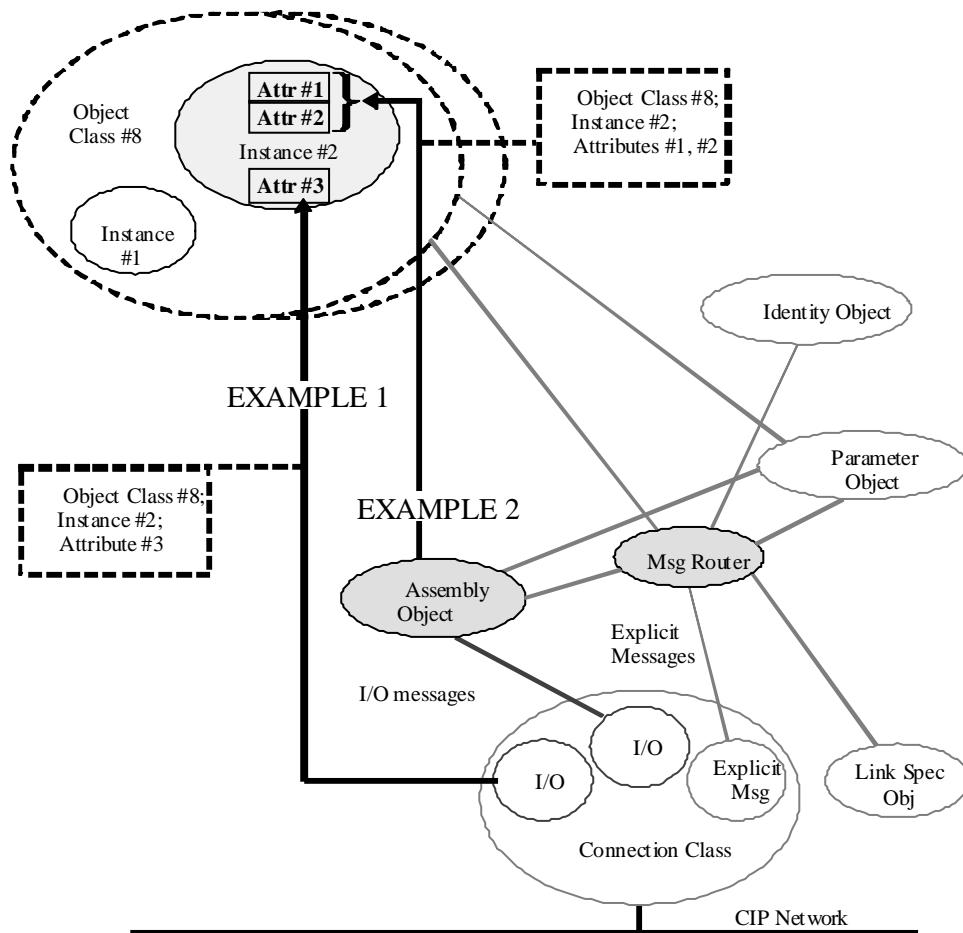
An Explicit Messaging Connection is capable of accessing any externally accessible object, including Application Objects. Explicit Messages can be used to deliver various service requests to Application Objects, including the reading and writing of Application Object attributes.

When an Explicit Message is received, the Explicit Messaging Connection delivers the message information to the Message Router. The Message Router delivers the message information to the appropriate *handler*. In the case of an Explicit Request Message, the *handler* is the target object specified in the request. In the case of an Explicit Response Message, the *handler* is the Client Application that previously issued the associated request.

#### 4-8.2 Access Through I/O Connections

I/O Connections contain attributes that *point to* or reference the Application Object(s) to which they deliver received data and/or from which they obtain data to transmit.

Figure 4-8.2 Access Paths to Application Object Data via I/O Connections



Application Objects can be *directly* referenced by an I/O Connection. This is illustrated by **EXAMPLE 1** in Figure 4-8.2. In this example, Attribute #3 within Application Object Class #8/Instance #2 (*Att #3*), is being directly accessed by an I/O Connection. If this reference was made by the **produced\_connection\_path** attribute of the I/O Connection, then *Att #3* data would be the information being produced by the I/O Connection. If this reference was made by the **consumed\_connection\_path** attribute of the I/O Connection, then *Att #3* would be the recipient of the data that is consumed by the I/O Connection.

It is possible to access multiple attributes over a single I/O Connection. This is accomplished using a special Application Object called the **Assembly Object**. With respect to data exchange over an I/O Connection, an Assembly Object performs the following:

- Assembles separate pieces of Class, Instance, and/or Attribute data together so they can be produced by a single I/O Connection.
- Receives data from an I/O Connection that is associated with separate Classes, Instances, and/or Attributes and distributes the data accordingly.

With this in mind, Assembly Objects provide an *indirect* reference to various data items. **EXAMPLE 2** in Figure 4-8.2 illustrates an Assembly Object that groups Attributes #1 and #2 of Class #8/Instance #2 so they can be accessed by a single I/O Connection. Note that an Assembly Object is capable of referencing items within more than 1 Object Class. See CIP Common, Chapter 5-5 for a detailed definition of the Assembly Class.

## 4-9 Extending Existing Objects and Defining New Objects

If your application requires a function that existing objects do not implement, you can:

- Add vendor-specific attributes to existing objects, or propose a standardized open addition to existing objects;
- Define a new open object or a new vendor-specific object (published or unpublished) to accommodate a new or revised functionality

Use the decision table below to determine how you would like to add functionality to your device.

An Object Class definition must indicate any special provisions that have been made with regards to the external access of its associated data.

**Table 4-9.1 How to Add Functionality**

If you	And	Then
want to add attributes and/or services to an existing object to extend its functionality,	you want this extension to be standardized,	propose an Open Extension.
want to add attributes and/or services to an existing object to extend its functionality,	you want to provide user-access to the extension,	create and publish a Vendor-specific Extension.
want to add attributes and/or services to an existing object to extend its functionality,	you want to keep the extension proprietary,	create, but do NOT publish Vendor-specific Extension.
want to create a device that has a function outside the realm of existing objects,	you want this extension to be standardized,	propose a new Open Object.
want to create a device that has a function outside the realm of existing objects,	you want to provide user-access to the extension,	create and publish a new Vendor-specific Object.
want to create a device that has a function outside the realm of existing objects,	you want to keep the new object proprietary,	create but do NOT publish a new Vendor-specific Object.

After determining whether you want to extend an existing object or define a new one, refer to the following pages for the appropriate information.

**Table 4-9.2 Where To Go To Make Changes**

For information about:	Go to section:
CIP Object Address Ranges	4-9.1
Making Extensions to Objects	4-9.2
Vendor-specific Extensions	4-9.2.1
Open Extensions	4-9.2.2
Defining a New Object	4-9.3
Vendor-specific Object	4-9.3.1
Open Object	4-9.3.2
Defining a New Common Service	4-9.4

#### 4-9.1 CIP Object Address Ranges

Whether you decide to extend existing open objects or define new objects, you must know the CIP-defined Object Addressing Ranges. There are three categories:

**Table 4-9.3 Object Address Range Categories**

This range	Refers to
Open	A value available to all CIP participants. Open values are defined in the CIP Common Specification and the associated network companion specifications. All Object Classes and services defined in CIP Specification fall under this category.
Vendor-specific	A range of values specific to the vendor of a device. These are used by vendors to extend their devices beyond the available <i>Open</i> options. A vendor internally manages the use of values within this range. Applies to object classes, attributes, and services.
Object-class-specific	A range of values whose meaning is defined by an Object Class. This range applies to Service Code definitions.

Table 4-9.4 defines the ranges applicable to Class ID values.

**Table 4-9.4 Class ID**

Range	Meaning	Quantity
00 - 63 <sub>hex</sub>	Open	100
64 <sub>hex</sub> - C7 <sub>hex</sub>	Vendor Specific	100
C8 <sub>hex</sub> - EF <sub>hex</sub>	Reserved by CIP for future use	40
F0 <sub>hex</sub> - 2FF <sub>hex</sub>	Open	528
300 <sub>hex</sub> - 4FF <sub>hex</sub>	Vendor Specific	512
500 <sub>hex</sub> - FFFF <sub>hex</sub>	Reserved by CIP for future use	64,256

Table 4-9.5 defines the ranges applicable to Attribute ID values.

**Table 4-9.5 Attribute ID Range**

Range	Meaning	Quantity
00 - 63 <sub>hex</sub>	Open	100
64 <sub>hex</sub> - C7 <sub>hex</sub>	Vendor Specific	100
C8 <sub>hex</sub> - FF <sub>hex</sub>	Reserved by CIP for future use	56
100 <sub>hex</sub> - 2FF <sub>hex</sub>	CIP Common	512
300 <sub>hex</sub> - 4FF <sub>hex</sub>	Vendor Specific	512
500 <sub>hex</sub> - 8FF <sub>hex</sub>	CIP Common	1024
900 <sub>hex</sub> - CFF <sub>hex</sub>	Vendor Specific	1024
D00 <sub>hex</sub> - FFFF <sub>hex</sub>	Reserved by ODVA/CI for future use	62208

Table 4-9.6 defines the ranges applicable to Service Code values.

**Table 4-9.6 Service Code Ranges**

Range	Meaning	Quantity
00 - 31 <sub>hex</sub>	Open. These are referred to as <i>CIP Common Services</i> . These are defined in Appendix A.	50
32 <sub>hex</sub> - 4A <sub>hex</sub>	Vendor Specific	25
4B <sub>hex</sub> - 63 <sub>hex</sub>	Object Class Specific	25
64 <sub>hex</sub> - 7F <sub>hex</sub>	Reserved by CIP for future use	28
80 <sub>hex</sub> - FF <sub>hex</sub>	Invalid/Not used	128

## 4-9.2 Making Extensions to Objects

If an existing open object provides functions that closely match those of your product but do not completely support an implementation, then you may want to extend the existing object instead of creating a new object.

You can modify the existing object to include a new behavior that would satisfy your desired function. A modification could entail simply adding state flags or diagnostics, or could mean inheriting behaviors of another existing object class in your product line.

There are two types of extensions to open objects:

- vendor specific
  - published (via user manual, EDS, etc)
  - unpublished
- open (extensions may be made only via actions in CIP Special Interest Groups)

An extension of an existing **open** object definition:

- can add new **open** or **vendor specific** attributes to the most recent definition. Deletion of **open** attributes is not allowed. **Open attributes can be obsoleted, but obsoleted attributes shall still be shown in the object definition and marked as obsolete.**
- can only add **open object attribute data** to the end of the Get\_Attributes\_All and the Set\_Attributes\_All structure definitions via actions in CIP Special Interest Groups.
- when extending an object, if the behavior of the existing public attributes or public services will change, the public revision of the object shall be incremented.
- can add **open** or **vendor specific services** to the list of services. Deletion of **open** services is not allowed. **Open services can be obsoleted, but obsoleted services shall still be shown in the object definition and marked as obsolete.**

### 4-9.2.1 Vendor-specific Extensions

If you want to add attributes and/or services to an existing object to extend its functionality, and you want only your users to have access to the extension, then create a vendor-specific extension.

**Important:** Be sure to publish your vendor-specific extension in some way for your users. If you don't publish the extension, then it is of little value because your users cannot access the added capability produced by the extension.

If you want other CIP participants to standardize on your extension, then propose an open extension (see Section 4-9.2.2) to the ODVA/CI organizations and wait for approval.

Regardless of whether you choose to keep the extension vendor-specific or propose it as an open extension, the process for defining an extension is the same.

Use the following example to examine the necessary steps for creating a vendor-specific, extension.

#### **4-9.2.1.1 Example**

For example purposes, assume you have a *Widget Object*, which is an existing open object that has two Class attributes and three Instance Attributes.

**Table 4-9.7 Class Attributes**

Number	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1	Get	Revision	UINT	Revision of this object	
2	Get	Max Instance	UINT	Maximum Instance Number	

**Table 4-9.8 Instance Attributes**

Number	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1	Get	Number of attributes	USINT	Number supported in this product	
2	Set	Output Value	BOOL		0 = off; 1 = on
3	Get	Status	UINT	Current status of this object instance	

This object supports these Common Services:

- Get\_Attributes\_All
- Set\_Attributes\_All

These services specify the following attributes:

**Table 4-9.9 Attributes Specified by Object Services**

Service	Attributes
Get_Attributes_All	1, 2, 3
Set_Attributes_All	2

In addition to Common Services, the Widget Object in this example supports these Object Specific Services:

- Upload\_Widget\_Attributes\_All
- Download\_Widget\_Attributes\_All

The current capability of the Widget Object closely matches the function of your product. However, it does not completely support your desired implementation. You want to extend the object and then publish the extension for your users.

For example, this hypothetical device requires the instance attributes Output Value, Status, and Number of Attributes, but needs an additional attribute specifying *Input Value*, which you would like to publish to your users.

To extend the existing Widget Object to include the proprietary *Input Value* attribute:

1. **Add an attribute(s):** Because this is a vendor specific attribute it needs to be in the vendor specific attribute value range. In this example, 100 has been chosen as the attribute number.

The new list of Instance Attributes looks like this:

**Table 4-9.10 Instance Attributes for Extended Widget Object**

Number	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1	Get	Number of attributes	USINT	Number supported in this product	
2	Set	Output Value	BOOL		0 = off; 1 = on
3	Get	Status	UINT	Current status of this object instance	
<b>100</b>	<b>Set</b>	<b>Input Value</b>	<b>BOOL</b>		<b>0 = off; 1 = on</b>

2. Because this attribute is a vendor specific attribute the Set\_Attributes\_All and Get\_Attributes\_All service definitions are not changed (vendor specific attributes are not included in Set\_Attributes\_All and Get\_Attributes\_All).
3. Add to list of object specific services, if necessary.
4. Specify the new behavior with a new version of the State Transition Diagram and necessary text.

#### 4-9.2.1.2 Implementing a Vendor-specific Extension

Whether adding vendor-specific attributes that are published for your users or proposed as a standard, you must plan for differences in:

- Revisions
- Get\_Attributes\_All service response
- Set\_Attributes\_All service request

##### 4-9.2.1.2.1 Revisions

Every object has a Class Attribute called “Revision.” The Revision of an object specifies the interface to that object, which encompasses all of the items in the object specification, including services, attributes, connections and behavior.

If the value of the Revision attribute is 01, then support of the Revision attribute is OPTIONAL in the object’s implementation. If the value is greater than 01, then support of the Revision attribute is REQUIRED.

**Table 4-9.11 Class Attributes**

Number	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1	Get	Revision	UINT	Revision of this object	
2	Get	Max Instance	UINT	Maximum Instance Number	

**Important:** If you extend an object in a Server device but not in a potential Client, then the Revision of the object in the Server device is different from the revision expected by the Client. As a result, the Client will not recognize the extension, and will not implement its new behavior.

To remedy this situation, you must document differences between the original object definition and the revised object definition. You may either:

- Update the Client; or
- Provide for the difference in revisions by allowing the Client to process error codes from the Server. You can program the Client to do one of the following:
  - Report the revision mismatch with an error handler that verifies the object revision; or
  - Determine the revision of the Server's object with an error handler that also performs special-case processing.

**Important:** The Revision class attribute is not the revision of an implementation (which is reflected in the Identity Object, Minor/Major revision status bits), but the revision of the **open object definition**.

If you want to specify a revision of the vendor-specific object definition, we recommend defining a “vendor-specific revision” class attribute in the vendor-specific address range.

The unrecognized attributes are either *critical* or *non-critical* to the behavior of the object.

**Table 4-9.12 Unrecognized Attribute Handling**

If the unrecognized attributes are	Then
Critical	Processing the service results in an error. Client must be prepared to process the error.
Non-critical	the Server processes only the open attributes and ignores the vendor-specific attributes (which are initialized to default values).

#### 4-9.2.2 Open Extensions

If you want to add attributes and/or services to an existing object to extend its functionality, and you want this extension to be standardized, then create an open extension in the same way you would a vendor-specific extension. See Steps 1 through 4 in Section 4-9.2.1.1, Example, and adhere to the following exceptions:

- Propose the use of “Open” attributes using the open ID range instead of “Vendor-specific.”

- Submit the extension to ODVA/CI for approval

**Important:** You cannot ship product based on your proposed extension until it is approved.

#### **4.9.3 Defining a New Object**

When defining a new object, either vendor-specific or open, follow these general guidelines:

- **Break large objects into smaller objects:** For example, break a “Controller” into “Data Table”, which can be read and written, and “Program”, which can be edited.
- **Avoid multiple interdependencies:** Don’t break an object into multiple objects if this creates many interdependencies. Keep things that are tightly coupled in one object.
- **Hide some information:** Hide those aspects that are not important to the object being specified. For example, if you can store a set of data as an array or in a linked list, don’t specify one or the other. Leave the set of data as “data storage”.
- **Do not hide important semantics:** Don’t hide aspects that are important to the object being specified. For example, don’t write a value to a special location to cause a mode change. Use an interface that allows you to clearly state the desired action.
- **Generalize:** Ask, “If product A needs an object from product B, will it also be needed in product C? If this is a possibility, how can I define the object so that it is also useful in product C?”
- **Provide for expansion:** When possible, use variable length fields. You can extend fields in the future to handle cases not considered today.

**Important:** It is preferable to extend an existing object than to define a new one. Fewer larger objects are more convenient than a proliferation of smaller objects that have similarities.

In addition, use the numeric value itself instead of encoding the numeric value into a smaller number of bits, so that “unencodable” values become necessary in the future.

- **Be clear:** Keep all statements, references and terms as clear and obvious as possible by using meaningful names. Don’t add complexity where not needed.
- **Make an object widely applicable:** Make sure the object is useful in a wide range of products (i.e., from a proximity sensor to a motor starter).

##### **4.9.3.1 Vendor-specific**

If you want your CIP-compatible product to perform a function that existing open objects cannot implement, then you may define a Vendor-specific Object within the appropriate address ranges. See Table 4-9.13.

**Table 4-9.13 Vendor-specific Object Address Ranges**

Type	Range	Quantity
Class ID	64 <sub>hex</sub> - C7 <sub>hex</sub>	100
	300 <sub>hex</sub> - 4FF <sub>hex</sub>	512
Attribute ID – Vendor-Specific	64 <sub>hex</sub> - C7 <sub>hex</sub>	100
	300 <sub>hex</sub> - 4FF <sub>hex</sub>	512
	900 <sub>hex</sub> - CFF <sub>hex</sub>	1024
Service Code – Vendor-Specific	32 <sub>hex</sub> - 4A <sub>hex</sub>	25

Service Code – Object-Specific	4B <sub>hex</sub> - 63 <sub>hex</sub>	25
--------------------------------	---------------------------------------	----

You may choose to create a vendor-specific object for one or more of the following reasons:

- You want to create a device that has a function outside the realm of existing objects.
- An existing product does not fit the function of existing objects.
- You want to create an object that has proprietary attributes and behavior.

Defining vendor-specific objects within specified Class ID, Attribute ID, and Service Code values has the following advantages and disadvantages:

**Table 4-9.14 Advantages and Disadvantages of Vendor-specific Objects**

Advantages	Disadvantages
Vendor controls the object	The vendor must manage the dissemination of the object definition to other parties.
Innovation not stifled or slowed by standards process.	The Client must verify that the target device is of the vendor type that includes this new object definition.

#### 4-9.3.2 Open

If you think a defined object and its behavior should be standardized, then propose an object specification with the Class Code and attribute IDs in the CIP Open address ranges to ODVA/CI. Also, specify common services used. See Table 4-9.15.

**Table 4-9.15 Open Address Ranges**

Type	Range	Quantity
Class ID	00 - 63 <sub>hex</sub>	100
	F0 <sub>hex</sub> - 2FF <sub>hex</sub>	528
Attribute ID	00 - 63 <sub>hex</sub>	100
	100 <sub>hex</sub> - 2FF <sub>hex</sub>	512
	500 <sub>hex</sub> - 8FF <sub>hex</sub>	1024
Service Code – Open	00 - 31 <sub>hex</sub>	50

You may follow the object template used in this chapter as well as the objects defined in Chapter 5, CIP Object Library. This template includes:

- Class code in open range
- Description of the object
- Class attributes
- Instance attributes
- Common services supported
- Object-specific services supported
- Connections
- Behavior

Submit the object proposal to one of the Joint Special Interest Groups (JSIGs) in ODVA/CI for approval.

#### **4-9.4      New Common Service**

If you want to define a new service that extends the current list of CIP Common Services, propose the service (in the open service code range) to ODVA/CI.

This page is intentionally left blank

## **Volume 1: Common Industrial Protocol (CIP<sup>TM</sup>) Specification**

### **Chapter 5: Object Library**

---

## Contents

5-1	Object Specifications .....	21
5-2	Identity Object .....	23
5-2.1	Class Attributes .....	23
5-2.2	Instance Attributes .....	23
5-2.2.1	Semantics .....	25
5-2.3	Common Services .....	29
5-2.3.1	Reset Service .....	29
5-2.3.2	Get_Attributes_All Response .....	30
5-2.4	Object-specific Services .....	32
5-2.5	Object-specific General and Extended Status Codes .....	32
5-2.6	Behavior .....	33
5-3	Message Router Object .....	37
5-3.1	Class Attributes .....	37
5-3.2	Instance Attributes .....	37
5-3.3	Common Services .....	37
5-3.3.1	Get_Attributes_All Response .....	38
5-3.4	Object-Specific Services .....	39
5-3.4.1	Symbolic Translation .....	39
5-3.5	Behavior .....	41
5-3.5.1	Service Request .....	41
5-3.5.2	Service Response .....	41
5-4	DeviceNet Object .....	42
5-5	Assembly Object .....	43
5-5.1	Class Attributes .....	44
5-5.2	Instance Attributes .....	44
5-5.3	Common Services .....	45
5-5.4	Object-specific Services .....	46
5-5.5	Behavior .....	46
5-5.5.1	Static Assemblies .....	47
5-5.5.2	Dynamic Assemblies .....	48
5-5.5.3	Connection Points .....	49
5-6	Connection Object .....	50
5-7	Connection Manager Object .....	51
5-8	Register Object .....	52
5-8.1	Class Attributes .....	52
5-8.2	Instance Attributes .....	52
5-8.3	Common Services .....	52
5-8.4	Object-specific Services .....	53
5-8.5	Behavior .....	53
5-9	Discrete Input Point Object .....	54
5-9.1	Revision History .....	54
5-9.2	Class Attributes .....	54
5-9.3	Instance Attributes .....	55
5-9.4	Common Services .....	55
5-9.4.1	Get_Attributes_All Response .....	56
5-9.4.2	Set_Attributes_All Request .....	57
5-9.5	Object-specific Services .....	57
5-9.6	Behavior .....	57
5-9.6.1	Attribute Access Rules .....	60
5-10	Discrete Output Point Object .....	61
5-10.1	Class Attributes .....	61
5-10.2	Instance Attributes .....	61
5-10.2.1	Value .....	63

5-10.2.2	Status.....	63
5-10.2.3	Fault and Idle Attributes .....	63
5-10.2.4	Run_Idle Command .....	64
5-10.2.5	Flash.....	64
5-10.2.6	Flash Rate.....	64
5-10.3	Common Services .....	64
5-10.3.1	Get_Attributes_All Response.....	65
5-10.3.2	Set_Attributes_All Request.....	66
5-10.4	Object-specific Services .....	67
5-10.5	Behavior.....	67
5-10.5.1	Attribute Access.....	70
5-11	Analog Input Point Object.....	71
5-11.1	Revision History .....	71
5-11.2	Class Attributes.....	71
5-11.3	Instance Attributes .....	72
5-11.4	Common Services .....	73
5-11.4.1	Get_Attributes_All Response.....	73
5-11.4.2	Set_Attributes_All Request.....	74
5-11.5	Object-specific Services .....	75
5-11.6	Behavior.....	75
5-11.6.1	Attribute Access Rules.....	77
5-12	Analog Output Point Object.....	79
5-12.1	Class Attributes.....	79
5-12.2	Instance Attributes .....	80
5-12.3	Common Services .....	82
5-12.3.1	Get_Attributes_All Response.....	82
5-12.3.2	Set_Attributes_All Request.....	83
5-12.4	Object-specific Services .....	84
5-12.5	Behavior.....	84
5-12.5.1	Attribute Access Rules.....	88
5-13	Presence Sensing Object.....	90
5-13.1	Class Attributes.....	90
5-13.2	Instance Attributes .....	90
5-13.2.1	Semantics .....	91
5-13.3	Common Services .....	92
5-13.4	Object-specific Services .....	92
5-13.5	Behavior.....	93
5-14	Parameter Object.....	94
5-14.1	Class Attributes.....	95
5-14.1.1	Semantics .....	95
5-14.2	Instance Attributes .....	97
5-14.2.1	Semantics .....	99
5-14.3	Common Services .....	103
5-14.3.1	Get_Attributes_All Response.....	104
5-14.4	Object-specific Services .....	106
5-14.5	Behavior.....	107
5-15	Parameter Group Object .....	109
5-15.1	Class Attributes.....	109
5-15.1.1	Semantics .....	109
5-15.2	Instance Attributes, Abbreviated Static Set .....	110
5-15.3	Instance Attributes, Extended Static Set .....	110
5-15.4	Common Services .....	111
5-15.4.1	Get_Attributes_All Response.....	111
5-15.5	Object-specific Services .....	112
5-15.6	Behavior.....	112

5-16	Group Object .....	113
5-16.1	Class Attributes .....	113
5-16.2	Instance Attributes .....	114
5-16.3	Common Services .....	114
5-16.3.1	Get_Attributes_All Response .....	114
5-16.4	Object-specific Services .....	116
5-16.5	Behavior .....	116
5-16.5.1	Attribute Access Rules .....	117
5-16.5.1.1	Status Attribute .....	117
5-16.5.1.2	Owner Vendor ID, Owner Serial Number .....	117
5-17	Discrete Input Group Object .....	118
5-17.1	Class Attributes .....	118
5-17.2	Instance Attributes .....	119
5-17.3	Semantics .....	119
5-17.3.1	On_OffDelay, Off_OnDelay .....	119
5-17.4	Common Services .....	119
5-17.4.1	Get_Attributes_All Response .....	120
5-17.4.2	Set_Attributes_All Request .....	121
5-17.5	Object-specific Services .....	121
5-17.6	Behavior .....	121
5-17.6.1	Attribute Access Rules .....	122
5-17.6.1.1	Status Attribute .....	122
5-18	Discrete Output Group Object .....	123
5-18.1	Class Attributes .....	123
5-18.2	Instance Attributes .....	124
5-18.3	Semantics .....	124
5-18.3.1	Command, Fault Action, Fault Value, Idle Action, Idle Value .....	124
5-18.4	Common Services .....	125
5-18.4.1	Get_Attributes_All Response .....	125
5-18.4.2	Set_Attributes_All Request .....	127
5-18.5	Object-specific Services .....	127
5-18.6	Behavior .....	127
5-18.6.1	Attribute Access Rules .....	128
5-18.6.1.1	Status Attribute .....	128
5-19	Discrete Group Object .....	129
5-19.1	Class Attributes .....	129
5-19.2	Instance Attributes .....	130
5-19.3	Common Services .....	130
5-19.3.1	Get_Attributes_All Response .....	130
5-19.4	Object-specific Services .....	131
5-19.5	Behavior .....	132
5-19.5.1	Attribute Access Rules .....	132
5-19.5.1.1	Status Attribute .....	132
5-20	Analog Input Group Object .....	133
5-20.1	Class Attributes .....	133
5-20.2	Instance Attributes .....	134
5-20.3	Semantics .....	134
5-20.3.1	Value Data Type .....	134
5-20.4	Common Services .....	135
5-20.4.1	Get_Attributes_All Response .....	135
5-20.4.2	Set_Attributes_All Request .....	136
5-20.5	Object-specific Services .....	137
5-20.6	Behavior .....	137
5-20.6.1	Attribute Access Rules .....	138
5-20.6.1.1	Status Attribute .....	138

5-20.6.1.2	Value Data Type Attribute .....	138
5-20.6.1.3	Owner Vendor ID and Owner Serial Number Attributes .....	138
5-20.6.1.4	Temp Mode Attribute .....	138
5-21	Analog Output Group Object.....	139
5-21.1	Class Attributes .....	139
5-21.2	Instance Attributes .....	140
5-21.3	Semantics .....	141
5-21.3.1	Command, Fault Action, Fault Value, Idle Action, Idle Value, Value Data Type .....	141
5-21.3.2	Value Data Type .....	141
5-21.4	Common Services .....	141
5-21.4.1	Get_Attributes_All Response .....	142
5-21.4.2	Set_Attributes_All Request.....	143
5-21.5	Object-specific Services .....	143
5-21.6	Behavior.....	143
5-21.6.1	Attribute Access Rules.....	144
5-21.6.1.1	Status Attribute .....	144
5-21.6.1.2	Value Data Type Attribute .....	144
5-21.6.1.3	Owner Vendor ID and Owner Serial Number Attributes .....	145
5-21.6.1.4	Fault Action/Idle Action Values for Analog Output Group and Analog Output Point .....	145
5-22	Analog Group Object.....	146
5-22.1	Class Attributes .....	146
5-22.2	Instance Attributes .....	147
5-22.3	Common Services .....	147
5-22.3.1	Get_Attributes_All Response .....	148
5-22.4	Object-specific Services .....	149
5-22.5	Behavior.....	149
5-22.5.1	Attribute Access Rules.....	150
5-22.5.1.1	Status Attribute .....	150
5-22.5.1.2	Value Data Type Attribute .....	150
5-22.5.1.3	Owner Vendor ID and Owner Serial Number Attributes .....	150
5-23	Position Sensor Object.....	151
5-23.1	Revision History .....	151
5-23.2	Class Attributes .....	151
5-23.3	Instance Attributes .....	151
5-23.4	Semantics .....	155
5-23.4.1	Position Value Unsigned - Attribute 3 .....	155
5-23.4.2	Position Value Signed - Attribute 10 .....	155
5-23.4.3	Value Bit Resolution – Attribute 5.....	155
5-23.4.4	Zero Offset – Attribute 6.....	156
5-23.4.4.1	CAM, CAM Low Limit, CAM High Limit – Attributes 4, 7 and 8.....	156
5-23.4.5	Auto Zero – Attribute 9.....	157
5-23.4.6	Position Sensor Type – Attribute 11 .....	157
5-23.4.7	Direction Counting Toggle – Attribute 12 .....	157
5-23.4.8	Commissioning Diagnostic Control – Attribute 13.....	157
5-23.4.9	Scaling Function Control – Attribute 14.....	158
5-23.4.10	Position Format – Attribute 15.....	158
5-23.4.11	Measuring Units per Span – Attribute 16 .....	159
5-23.4.12	Total Measuring Range in Measuring Units – Attribute 17.....	159
5-23.4.13	Position measuring increment – Attribute 18.....	159
5-23.4.14	Preset Value – Attribute 19 .....	159
5-23.4.15	COS Delta – Attribute 20.....	159
5-23.4.16	Position State Register, Position limits – Attributes 21 to 23 .....	160
5-23.4.17	Velocity Format– Attribute 25 .....	160
5-23.4.18	Velocity Resolution – Attribute 26 .....	160
5-23.4.19	Minimum Velocity / Maximum Velocity - Attribute 27 thru 28.....	160

5-23.4.20	Acceleration Value - Attribute 29 .....	160
5-23.4.21	Acceleration Format – Attribute 30 .....	161
5-23.4.22	Acceleration Resolution – Attribute 31.....	161
5-23.4.23	Minimum Acceleration / Maximum Acceleration - Attribute 32 thru 33 .....	161
5-23.4.24	Number of CAM Channels – Attribute 34.....	161
5-23.4.25	CAM Channel State Register – Attribute 35.....	163
5-23.4.26	CAM Channel Polarity Register – Attribute 36.....	163
5-23.4.27	CAM Channel Enable Register – Attribute 37.....	164
5-23.4.28	CAM Low Limit – Attribute 38 .....	164
5-23.4.29	CAM High Limit – Attribute 39 .....	164
5-23.4.30	CAM Hysteresis – Attribute 40.....	164
5-23.4.31	Operating Status – Attribute 41 .....	164
5-23.4.32	Physical Resolution – Attribute 42 .....	164
5-23.4.33	Number of Spans – Attribute 43 .....	165
5-23.4.34	Alarms – Attribute 44 .....	165
5-23.4.35	Supported Alarms – Attribute 45 .....	165
5-23.4.36	Alarm Flag – Attribute 46.....	165
5-23.4.37	Warnings – Attribute 47.....	165
5-23.4.38	Supported Warnings – Attribute 48 .....	166
5-23.4.39	Warning flag – Attribute 49 .....	166
5-23.4.40	Operating Time – Attribute 50.....	166
5-23.4.41	Offset Value – Attribute 51.....	166
5-23.5	Common Services .....	167
5-23.6	Object-specific Services .....	168
5-23.7	Behavior .....	168
5-24	Position Controller Supervisor Object .....	171
5-24.1	Revision History .....	171
5-24.2	Class Attributes .....	171
5-24.2.1	Consumed Axis Selection Number .....	171
5-24.2.2	Produced Axis Selection Number .....	172
5-24.3	Object Instance Attributes.....	172
5-24.3.1	Supervisor Attributes .....	172
5-24.3.2	Home and Index Attributes .....	173
5-24.3.3	Registration Attributes .....	174
5-24.3.4	Axis Following Attributes.....	175
5-24.4	Common Services .....	175
5-24.5	Object-specific Services.....	175
5-24.6	Behavior .....	175
5-24.6.1	Position Controller Supervisor State Diagrams .....	176
5-24.6.1.1	Home Input State Diagrams.....	176
5-24.6.1.2	Index Input State Diagram.....	177
5-24.6.1.3	Registration Input State Diagram.....	178
5-25	Position Controller Object .....	179
5-25.1	Revision History .....	179
5-25.2	Class Attributes .....	179
5-25.3	Instance Attributes .....	180
5-25.3.1	Semantics .....	184
5-25.3.1.1	Profile in Progress.....	184
5-25.3.1.2	Load Data Complete .....	185
5-25.4	Common Services .....	185
5-25.5	Object-specific Services.....	185
5-25.6	Behavior .....	185
5-25.6.1	Position Controller State Diagrams.....	186
5-25.6.1.1	Profile Move Generation State Diagrams .....	186
5-25.6.1.2	Servo Output Generation State Diagram .....	187

5-25.6.1.3	Torque Mode Output State Diagram.....	188
5-26	Block Sequencer Object.....	189
5-26.1	Class Attributes.....	189
5-26.2	Instance Attributes .....	189
5-26.3	Common Services .....	190
5-26.4	Object-specific Services.....	190
5-26.5	Behavior.....	190
5-26.5.1	Block Sequencer State Diagrams .....	190
5-27	Command Block Object.....	191
5-27.1	Class Attributes.....	191
5-27.2	Attributes .....	191
5-27.3	Command Specific Attribute Services .....	192
5-27.3.1	Modify Attribute Command - 01 .....	192
5-27.3.2	Wait Equals Command - 02 .....	192
5-27.3.3	Conditional Link Greater Than Command - 03 .....	193
5-27.3.4	Conditional Link Less Than Command - 04.....	194
5-27.3.5	Decrement Counter Command - 05 .....	194
5-27.3.6	Delay Command - 06 .....	194
5-27.3.7	Trajectory Command - 07 .....	195
5-27.3.8	Trajectory Command and Wait - 08.....	195
5-27.3.9	Velocity Change Command - 09 .....	195
5-27.3.10	Goto Home Command - 10.....	195
5-27.3.11	Goto Index Command - 11.....	196
5-27.3.12	Goto Registration Position Command - 12 .....	196
5-27.3.13	Common Services .....	196
5-28	Motor Data Object .....	197
5-28.1	Class Attributes.....	197
5-28.2	Instance Attributes .....	197
5-28.2.1	Motor Type Specific Motor Data Instance Attributes.....	198
5-28.2.1.1	AC Motor Instance Attributes.....	198
5-28.2.1.2	DC Motor Instance Attributes.....	199
5-28.3	Common Services .....	200
5-28.4	Object-specific Services .....	201
5-28.5	Behavior.....	201
5-29	Control Supervisor Object .....	203
5-29.1	Class Attributes.....	203
5-29.2	Instance Attributes .....	203
5-29.2.1	Operation Mode Values .....	205
5-29.2.2	NetFaultMode Attribute .....	205
5-29.2.3	NetIdleMode Attribute .....	206
5-29.2.4	ProtectMode Attribute.....	206
5-29.2.5	Fault/Warning Code Style Attribute .....	206
5-29.3	Common Services .....	207
5-29.4	Object Specific Services .....	207
5-29.5	Behavior.....	207
5-29.5.1	Run/Stop Event Matrix .....	209
5-29.6	Fault and Warning codes .....	209
5-29.7	Object-specific General and Extended Status Codes .....	215
5-30	AC/DC Drive Object .....	216
5-30.1	Class Attributes.....	216
5-30.2	Instance Attributes .....	216
5-30.2.1	Semantics .....	220
5-30.2.1.1	Speed Control and Speed Status .....	220
5-30.3	Common Services .....	220

5-30.4	Object-specific Services.....	221
5-30.5	Behavior .....	221
5-30.5.1	Enhanced Speed Control Behavior .....	224
5-30.5.2	Scaling of attribute values.....	224
5-31	Acknowledge Handler Object.....	226
5-31.1	Class Attributes .....	226
5-31.2	Class Services .....	226
5-31.3	Instance Attributes .....	227
5-31.4	Instance Services.....	228
5-31.5	Object-specific Services.....	228
5-31.6	Behavior and Configuration of Acknowledged Data Production .....	229
5-31.6.1	Acknowledged Data Production .....	229
5-31.6.2	Change of State Examples .....	230
5-31.6.2.1	Acknowledged Change of State (Using one connection object and one COS consumer).....	230
5-31.6.2.2	Acknowledged Change of State (Using multiple connection objects and COS consumers).....	231
5-31.6.3	Use of Timers with Acknowledged Data Production .....	232
5-32	Overload Object.....	237
5-32.1	Class Attributes .....	237
5-32.2	Instance Attributes .....	237
5-32.2.1	Scaling of attribute values .....	238
5-32.3	Common Services .....	238
5-32.4	Object-specific Services.....	238
5-32.5	Behavior .....	239
5-33	Softstart Object .....	240
5-33.1	Class Attributes .....	240
5-33.2	Instance Attributes .....	240
5-33.3	Common Services .....	241
5-33.4	Object-specific Services.....	241
5-33.5	Behavior.....	241
5-33.5.1	Starting Behavior .....	241
5-33.5.1.1	Voltage Ramp Starting.....	241
5-33.5.1.2	Current Limit Starting.....	242
5-33.5.1.3	Voltage Ramp With Current Limited Starting .....	242
5-33.5.2	Stopping Behavior .....	242
5-33.5.2.1	Coasting to Stop.....	243
5-33.5.2.2	Ramp to Stop .....	243
5-33.5.2.3	DC Brake to Stop.....	243
5-33.5.2.4	Vender Specific Stopping .....	243
5-33.5.3	Other Attribute Behaviors .....	243
5-33.5.3.1	Rotation (Attribute 11).....	243
5-33.5.3.2	EnergySaver (Attribute 14).....	243
5-33.5.4	Soft Starter Output Voltage Behavior Description .....	244
5-34	Selection Object.....	245
5-34.1	Class Attributes .....	245
5-34.2	Class Services .....	245
5-34.3	Instance Attributes .....	245
5-34.4	Semantics .....	247
5-34.4.1	State.....	247
5-34.4.2	max_destinations.....	247
5-34.4.3	number_of_destinations .....	247
5-34.4.4	destination_list .....	247
5-34.4.5	max_sources.....	247
5-34.4.6	number_of_sources .....	247
5-34.4.7	source_list .....	247
5-34.4.8	source_used.....	247

---

5-34.4.9	source_alarm .....	248
5-34.4.10	algorithm_type .....	248
5-34.4.11	detection_count .....	248
5-34.4.12	selection_period .....	248
5-34.4.13	Object Source List .....	248
5-34.4.14	Destination Used .....	248
5-34.4.15	Input Data Value .....	248
5-34.4.16	Output Data Value .....	249
5-34.5	Instance Services .....	249
5-34.6	Instance Behavior .....	249
5-34.6.1	Message Distribution .....	251
5-34.6.2	Message Selection .....	252
5-34.6.3	Message Selection Algorithms .....	252
5-34.6.3.1	Pass Through .....	252
5-34.6.3.2	Hot Backup .....	253
5-34.6.3.3	First Arrival .....	253
5-34.6.3.4	Last Arrival .....	255
5-34.6.3.5	Programmable Data Flow .....	255
5-34.6.4	Examples .....	256
5-35	S-Device Supervisor Object .....	259
5-35.1	Class Attributes .....	260
5-35.1.1	Subclasses .....	260
5-35.2	Instance Attributes .....	260
5-35.2.1	Subclasses .....	264
5-35.3	Semantics .....	264
5-35.3.1	Device Type .....	264
5-35.3.2	Manufacturer's Name .....	264
5-35.3.3	Software Revision Level .....	264
5-35.3.4	Hardware Revision Level .....	264
5-35.3.5	Manufacturer's Serial Number .....	265
5-35.3.6	Device Status .....	265
5-35.3.7	Exception Status .....	265
5-35.3.8	Exception Detail Alarm and Exception Detail Warning .....	266
5-35.3.9	Common Exception Detail .....	267
5-35.3.10	Device Exception Detail .....	267
5-35.3.11	Manufacturer Exception Detail .....	268
5-35.3.12	Alarm Enable and Warning Enable .....	268
5-35.3.13	Time .....	269
5-35.3.14	Scheduled Maintenance Expiration Timer .....	269
5-35.3.15	Scheduled Maintenance Expiration Warning Enable .....	269
5-35.3.16	Endpoint .....	269
5-35.3.17	Recipe .....	269
5-35.4	Common Services .....	270
5-35.5	Object-specific Services .....	270
5-35.5.1	Abort .....	270
5-35.5.2	Recover .....	270
5-35.5.3	Perform_Diagnostics .....	271
5-35.5.4	TestID Parameter .....	271
5-35.6	Behavior .....	271
5-35.6.1	S-Device Supervisor Object States .....	271
5-35.6.2	State Event Matrix .....	273
5-35.6.3	S-Device Supervisor and Identity Object Interface .....	274
5-35.6.4	S-Device Supervisor and Application Object Interface .....	274
5-35.7	S-Device Supervisor Instance Subclass 01 — Power Generator .....	274
5-35.7.1	Instance Attributes .....	275

5-35.7.2	Services .....	276
5-35.7.3	Behavior .....	276
5-35.7.3.1	Energy Control Application Process .....	276
5-36	S-Analog Sensor Object .....	277
5-36.1	Class Attributes .....	277
5-36.1.1	Subclasses .....	277
5-36.2	Instance Attributes .....	277
5-36.2.1	Subclasses .....	281
5-36.3	Semantics .....	282
5-36.3.1	Data Type .....	282
5-36.3.2	Data Units .....	282
5-36.3.3	Value, Offset (A and B) and Gain .....	282
5-36.3.4	Status .....	282
5-36.3.5	Trip Points, Hysteresis and Settling Time .....	283
5-36.3.6	Safe State .....	283
5-36.3.7	Safe Value .....	283
5-36.3.8	Autozero Enable and Autozero Status .....	283
5-36.3.9	Autorange Enable and Range Multiplier .....	284
5-36.3.10	Produce Trigger Delta .....	284
5-36.3.11	Calibration Object Instance .....	284
5-36.3.12	Produce Trigger Delta Type .....	285
5-36.3.13	Value Descriptor .....	285
5-36.4	Common Services .....	285
5-36.5	Object-specific Services .....	286
5-36.5.1	Zero_Adjust and Gain_Adjust Services .....	286
5-36.6	Behavior .....	286
5-36.6.1	Data Type .....	287
5-36.7	S-Analog Sensor Object Class Subclass 01 (Instance Selector) .....	288
5-36.7.1	Class Attributes .....	288
5-36.7.2	Semantics .....	288
5-36.7.2.1	Active Value .....	288
5-36.7.2.2	Active Instance Number .....	288
5-36.7.2.3	Number of Gauges .....	289
5-36.7.3	Services .....	289
5-36.7.4	Behavior .....	289
5-36.8	S-Analog Sensor Object Instance Subclass 01 (Flow Diagnostics) .....	290
5-36.8.1	Instance Attributes .....	290
5-36.8.2	Services .....	290
5-36.8.3	Behavior .....	290
5-36.8.3.1	Flow Totalizer and Flow Hours Process .....	290
5-36.9	S-Analog Sensor Object Instance Subclass 02 (Heat Transfer Vacuum Gauge) .....	291
5-36.9.1	Instance Attributes .....	291
5-36.9.2	Semantics .....	291
5-36.9.2.1	Cable Length .....	291
5-36.9.2.2	Sensor Temperature .....	291
5-36.9.2.3	Sensor Warning .....	292
5-36.9.2.4	Sensor Alarm .....	292
5-36.9.2.5	Status Extension .....	292
5-36.9.3	Services .....	292
5-36.9.4	Behavior .....	292
5-36.10	S-Analog Sensor Object Instance Subclass 03 (Diaphragm Gauge) .....	293
5-36.10.1	Instance Attributes .....	293
5-36.10.2	Semantics .....	293
5-36.10.2.1	Sensor Warning .....	293
5-36.10.2.2	Sensor Alarm .....	294

5-36.10.2.3	Status Extension.....	294
5-36.10.3	Services .....	294
5-36.10.4	Behavior .....	294
5-36.11	S-Analog Sensor Object Instance Subclass 04 (Cold Cathode Ion Gauge) .....	295
5-36.11.1	Instance Attributes .....	295
5-36.11.2	Semantics: .....	295
5-36.11.2.1	High Voltage.....	295
5-36.11.2.2	Sensitivity .....	295
5-36.11.2.3	Sensor Warning .....	296
5-36.11.2.4	Sensor Alarm.....	296
5-36.11.2.5	Status Extension.....	296
5-36.11.3	Services .....	296
5-36.11.3.1	Clear Overpressure High Voltage Off Alarm Service .....	297
5-36.11.3.2	Set High Voltage State Service.....	297
5-36.11.4	Behavior .....	297
5-36.11.4.1	Alarms/Warnings .....	297
5-36.11.4.2	No ignition detection .....	297
5-36.12	S-Analog Sensor Object Instance Subclass 05 (Hot Cathode Ion Gauge) .....	299
5-36.12.1	Instance Attributes .....	299
5-36.12.2	Semantics .....	301
5-36.12.2.1	Active Degas Filament.....	301
5-36.12.2.2	Degas Time Off Interval and Degas Time Off Remaining .....	301
5-36.12.2.3	Degas Time On Interval and Degas Time On Remaining .....	301
5-36.12.2.4	Active Filament.....	301
5-36.12.2.5	Sensitivity .....	302
5-36.12.2.6	Mode Filament Selection .....	302
5-36.12.2.7	Sensor Warning .....	303
5-36.12.2.8	Sensor Alarm .....	303
5-36.12.2.9	Status Extension.....	303
5-36.12.3	Services .....	304
5-36.12.3.1	Clear Emission Off Alarm Service .....	304
5-36.12.3.2	Set Emission State .....	304
5-36.12.3.3	Set Degas State .....	304
5-36.12.4	Behavior .....	305
5-36.12.4.1	Sensor Alarms/Warnings: Filaments .....	305
5-36.12.4.2	Sensor Alarm Byte 1:.....	305
5-36.12.4.3	Sensor Warning Byte 1:.....	306
5-36.12.5	S-Analog Sensor Object Sub-State for the Executing State.....	306
5-36.12.6	S-Analog Sensor Sub-State Event Matrix.....	307
5-36.13	S-Analog Sensor Object Instance Subclass 06 - Transfer Function .....	309
5-36.13.1	Instance Attributes .....	309
5-36.13.2	Semantics .....	309
5-36.13.2.1	Pressure Variable Mapping Function .....	309
5-37	S-Analog Actuator Object .....	310
5-37.1	Class Attributes .....	310
5-37.1.1	Subclasses .....	310
5-37.2	Instance Attributes .....	311
5-37.2.1	Subclasses .....	313
5-37.2.2	Semantics: .....	313
5-37.2.2.1	Data Type.....	313
5-37.2.2.2	Data Units .....	313
5-37.2.2.3	Value, Offset, Gain, Bias and Unity Gain Reference .....	314
5-37.2.2.4	Status.....	314
5-37.2.2.5	Trip Points and Hysteresis .....	314
5-37.2.2.6	Override .....	315

5-37.2.2.7	Safe State .....	315
5-37.2.2.8	Safe Value.....	315
5-37.3	Common Services .....	316
5-37.4	Object-Specific Services .....	316
5-37.5	Behavior.....	316
5-37.5.1	Data Type.....	316
5-38	S-Single Stage Controller Object.....	318
5-38.1	Class Attributes .....	318
5-38.1.1	Subclasses .....	318
5-38.2	Instance Attributes .....	319
5-38.2.1	Subclasses .....	321
5-38.3	Semantics: .....	321
5-38.3.1	Data Type.....	321
5-38.3.2	Data Units .....	321
5-38.3.3	Setpoint, Process Variable and Control Variable.....	321
5-38.3.4	Status.....	322
5-38.3.5	Control Mode .....	322
5-38.3.6	Safe State .....	322
5-38.3.7	Safe Value .....	323
5-38.4	Common Services .....	323
5-38.5	Object-specific Services .....	323
5-38.6	Behavior.....	323
5-38.6.1	Alarm and Warning Exception Conditions .....	323
5-38.6.2	Ramp Rate.....	324
5-38.6.3	Data Type.....	324
5-38.6.4	Control .....	324
5-38.7	S-Single Stage Controller Object Instance Subclass 01 - (PID and Source Select).....	325
5-38.7.1	Instance Attributes .....	325
5-38.7.2	Semantics .....	326
5-38.7.2.1	Calibrating State .....	326
5-38.7.2.2	Delay Time .....	326
5-38.7.2.3	Sensor Crossover High, Low .....	326
5-38.7.2.4	Expected Process Parameter .....	326
5-38.7.2.5	Kd,Ki,Kp .....	326
5-38.7.2.6	Control Direction .....	326
5-38.7.2.7	Process Variable Source .....	326
5-38.7.3	Services .....	327
5-38.7.3.1	Calibrate.....	328
5-38.7.4	Behavior.....	328
5-38.7.4.1	Process Variable Input Switching.....	329
5-38.7.4.2	Delay Time .....	329
5-38.7.4.3	Sensor Crossover Low .....	329
5-38.7.4.4	Sensor Crossover High .....	329
5-38.7.4.5	Control Direction .....	329
5-38.7.4.6	Process Variable Source .....	330
5-38.8	S-Single Stage Controller Instance Subclass 02 - DC Generator.....	331
5-38.8.1	Instance Attributes .....	331
5-38.8.2	Semantics .....	332
5-38.8.2.1	Output Max .....	332
5-38.8.2.2	Output Limit .....	332
5-38.8.2.3	Ramp Rate Increment .....	332
5-38.8.3	Services .....	332
5-38.8.4	Behavior.....	333
5-38.8.4.1	Pulse Application Process.....	333
5-38.9	S-Single Stage Controller Instance Subclass 03 - RF Generator .....	334

5-38.9.1	Instance Attributes .....	334
5-38.9.2	Semantics .....	335
5-38.9.2.1	Output Max .....	335
5-38.9.2.2	Output Limit .....	335
5-38.9.2.3	Ramp Rate Increment .....	335
5-38.9.3	Services .....	335
5-38.9.4	Behavior .....	335
5-38.9.4.1	Pulse Application Process .....	335
5-38.10	S-Single Stage Controller Instance Subclass 04 - Frequency Control .....	336
5-38.10.1	Instance Attributes .....	336
5-38.10.2	Semantics .....	336
5-38.10.2.1	Minimum, Maximum and Default Frequency Attributes .....	336
5-38.10.2.2	Frequency Limit High and Low Attributes .....	336
5-38.10.2.3	Setpoint .....	336
5-38.10.3	Services .....	337
5-38.10.4	Behavior .....	337
5-38.10.4.1	Frequency Control Application Process .....	337
5-38.10.4.2	Auto-tune Application Process .....	337
5-39	S-Gas Calibration Object .....	338
5-39.1	Class Attributes .....	339
5-39.1.1	Subclasses .....	339
5-39.2	Instance Attributes .....	339
5-39.2.1	Subclasses .....	341
5-39.3	Semantics: .....	341
5-39.3.1	Gas Standard Number .....	341
5-39.3.2	Valid Sensor Instances .....	341
5-39.3.3	Gas Symbol .....	341
5-39.3.4	Full Scale .....	341
5-39.3.5	Instance Application Example .....	342
5-39.4	Common Services .....	342
5-39.5	Object-specific Services .....	342
5-39.6	S-Gas Calibration Object Behavior .....	343
5-39.7	S-Gas Calibration Object Instance Subclass 01 (Standard T & P) .....	343
5-39.7.1	Instance Attributes .....	343
5-39.8	Services .....	343
5-39.9	Behavior .....	343
5-40	Trip Point Object .....	344
5-40.1	Class Attributes .....	344
5-40.1.1	Subclasses .....	344
5-40.2	Instance Attributes .....	344
5-40.2.1	Subclasses .....	346
5-40.3	Semantics .....	346
5-40.3.1	High/Low Trip Point, Enable and Status .....	346
5-40.3.2	Hysteresis and Delay .....	347
5-40.3.3	Data Type and Data Units .....	347
5-40.3.4	Polarity and Output .....	347
5-40.3.5	Source and Input .....	348
5-40.3.6	Destination and Output .....	348
5-40.4	Common Services .....	348
5-40.5	Object-Specific Services .....	348
5-40.6	Behavior .....	348
5-40.6.1	Trip Point Object States .....	348
5-40.6.2	Trip Point Object State Event Matrix .....	349
5-41	Drive Data Object .....	350
5-42	File Object .....	351

---

5-42.1	Class Attributes .....	351
5-42.2	Instance Attributes .....	352
5-42.2.1	Semantics .....	353
5-42.2.1.1	Instance_Name.....	353
5-42.2.1.2	Instance Format Version.....	353
5-42.2.1.3	File_Name.....	353
5-42.2.1.4	File Revision.....	353
5-42.2.1.5	File_Size .....	353
5-42.2.1.6	File Checksum .....	353
5-42.2.1.7	File Save Parameters.....	353
5-42.3	Common Services .....	354
5-42.3.1	Create Service Description.....	354
5-42.4	Object-specific Services.....	355
5-42.4.1	Initiate_Upload Service Description.....	355
5-42.4.2	Initiate_Download Service Description.....	355
5-42.4.3	Initiate_Partial_Read Service Description .....	356
5-42.4.4	Initiate_Partial_Write Service Description .....	357
5-42.4.5	Upload_Transfer Service Description.....	357
5-42.4.6	Download_Transfer Service Description.....	358
5-42.4.7	Clear File.....	359
5-42.5	Error Codes .....	359
5-42.5.1	Error codes for Initiate Services.....	359
5-42.5.2	Error codes for Transfer Services .....	360
5-42.6	Behavior.....	360
5-42.6.1	Checksum.....	360
5-42.6.2	State Event Matrix .....	362
5-42.7	Predefined File Instances .....	364
5-42.7.1	This Device's EDS File and Icon File – (C8 <sub>hex</sub> ) .....	365
5-42.7.2	Collection of Related EDS and Icon Files – (C9 <sub>hex</sub> ) .....	366
5-42.8	File Encoding Format .....	366
5-42.8.1	Table of File Encoding Format Values .....	366
5-42.8.2	Compressed File – ZLIB Encoding (01 <sub>hex</sub> ) .....	367
5-43	S-partial Pressure Object.....	368
5-43.1	Class Attributes .....	368
5-43.1.1	Class Level Subclasses.....	369
5-43.1.2	Semantics .....	369
5-43.1.2.1	Filament Control .....	369
5-43.1.2.2	Filament Status .....	370
5-43.2	Instance Attributes .....	370
5-43.2.1	Instance Level Subclasses .....	373
5-43.2.2	Semantics .....	373
5-43.2.2.1	Instance Type .....	373
5-43.2.2.2	AMU .....	373
5-43.2.2.3	Ending AMU .....	373
5-43.2.2.4	Gas Standard Number .....	374
5-43.2.2.5	Status.....	374
5-43.2.2.6	Instance Enable .....	374
5-43.2.2.7	Dwell Time and Electron Energy .....	374
5-43.2.2.8	Trip Points and Hysteresis .....	374
5-43.3	Common Services .....	375
5-43.3.1	Get_Attributes_All Response .....	375
5-43.4	Object-specific Services .....	376
5-43.4.1	Create_Range Service .....	376
5-43.4.2	Get_Instance_List Service .....	377
5-43.4.3	Get_Pressures Service.....	377

5-43.4.4	Get_All_Pressures Service.....	377
5-43.4.5	Group_Enable Service .....	378
5-43.5	Object Behavior .....	378
5-44	S-Sensor Calibration Object .....	379
5-44.1	Class Attributes.....	379
5-44.2	Subclasses .....	380
5-44.3	Instance Attributes .....	380
5-44.4	Subclasses .....	382
5-44.5	Semantics .....	382
5-44.5.1	Calibration ID Number .....	382
5-44.5.2	Valid Sensor Instances .....	382
5-44.5.3	Calibration Name .....	382
5-44.5.4	Full Scale .....	382
5-44.5.5	Temperature and Pressure.....	383
5-44.5.6	Instance Application Example .....	383
5-44.6	Common Services .....	383
5-44.7	Object-specific Services .....	383
5-44.7.1	Get_All_Instances Request Service Data Field Parameters.....	384
5-44.7.2	Get_All_Instances Success Response .....	384
5-44.8	S-Sensor Calibration Object Behavior .....	384
5-45	Event Log Object.....	385
5-45.1	Event Control .....	385
5-45.2	Class Attributes .....	386
5-45.2.1	Time Format.....	386
5-45.2.2	Present Time .....	387
5-45.2.3	Instance List .....	387
5-45.3	Instance Attributes .....	387
5-45.4	Semantics: .....	389
5-45.4.1	Attribute #1 - Instance Name .....	389
5-45.4.2	Attribute #2 - State .....	389
5-45.4.3	Attribute #3 - Event List Size .....	389
5-45.4.4	Attribute #4 - Event List .....	389
5-45.4.5	Attribute #5 - Event Enable .....	390
5-45.4.6	Attribute #6 - Event Silenced.....	390
5-45.4.7	Attribute #7 - Event State.....	390
5-45.4.8	Attribute #8 - Logged States Configuration.....	390
5-45.4.9	Attribute #9 - Logged Data Configuration.....	390
5-45.4.10	Attribute #10 - Log Full Action .....	391
5-45.4.11	Attribute #11 - Duplicate Event Action .....	391
5-45.4.12	Attribute #12 - Event/Data Log Maximum Size .....	391
5-45.4.13	Attribute #13 - Event/Data Log Size.....	391
5-45.4.14	Attribute #14 - Event/Data Log .....	391
5-45.4.14.1	Service Behavior:.....	392
5-45.4.15	Attribute #15 - Active Event List Size .....	394
5-45.4.16	Attribute #16 - Active Event List.....	394
5-45.4.17	Attribute #17 - Logged Event List Size .....	394
5-45.4.18	Attribute #18 - Logged Event List .....	394
5-45.4.19	Attribute #19 - Log Full .....	395
5-45.4.20	Attribute #20 - Log Not Empty .....	395
5-45.4.21	Attribute #21 - Log Overrun .....	395
5-45.4.22	Attribute #22 – Sequential Event/Data Access .....	395
5-45.4.23	Attribute #23 - Startup Behavior.....	396
5-45.4.24	Attribute #24 - Event Identifier Format .....	396
5-45.5	Common Services .....	397
5-45.5.1	Reset Service.....	397

5-45.6	Event/Data Log Instance Behavior .....	398
5-45.6.1	Event Behavior.....	400
5-46	Motion Axis Object.....	403
5-46.1	Introduction.....	403
5-46.2	Organization.....	403
5-46.2.1	Control Modes .....	403
5-46.2.2	Control Methods .....	403
5-46.2.3	Control Nomenclature.....	404
5-46.2.4	Position Control .....	404
5-46.2.5	Velocity Control.....	405
5-46.2.6	Acceleration Control .....	407
5-46.2.7	Torque Control.....	407
5-46.2.8	No Control .....	408
5-46.3	Drive Control Categories .....	409
5-46.4	Required vs. Optional in Implementation .....	409
5-46.5	Class Attributes .....	417
5-46.6	Semantics .....	421
5-46.6.1	Attribute #10 - Controller Consumed Connection Data.....	421
5-46.6.2	Attribute #11 - Controller Produced Connection Data.....	421
5-46.6.3	Attribute #14 – Node Control .....	422
5-46.6.4	Attribute #15 – Node Status.....	423
5-46.6.5	Attribute #16 – Node Faults.....	423
5-46.6.6	Attribute #17 – Node Alarms.....	424
5-46.7	Instance Attributes .....	424
5-46.8	Motion Control Configuration Attributes .....	425
5-46.8.1	Semantics .....	426
5-46.8.2	Attribute #1 – Control Mode.....	426
5-46.8.3	Attribute #2 – Control Method.....	426
5-46.9	Motor Attributes.....	427
5-46.9.1	General Motor Attributes .....	428
5-46.9.2	General PM Motor Attributes .....	431
5-46.9.3	General Rotary Motor Attributes .....	431
5-46.9.4	General Linear Motor Attributes.....	432
5-46.9.5	Rotary PM Motor Attributes .....	433
5-46.9.6	Linear PM Motor Attributes.....	433
5-46.9.7	Induction Motor Attributes .....	433
5-46.10	Feedback Attributes .....	434
5-46.10.1	Semantics .....	442
5-46.10.2	Attribute #3 – Feedback Configuration .....	442
5-46.11	Event Capture Attributes.....	442
5-46.11.1	Semantics .....	444
5-46.11.2	Attribute #256 – Event Checking Control .....	444
5-46.11.3	Attribute #257 – Event Checking Status.....	445
5-46.12	Command Reference Generation Attributes .....	446
5-46.12.1	Semantics .....	451
5-46.12.2	Attribute #301 – Interpolation Control .....	451
5-46.13	Control Mode Attributes .....	451
5-46.13.1	Position Loop Attributes .....	451
5-46.13.2	Velocity Loop Attributes .....	453
5-46.13.3	Acceleration Control Attributes.....	456
5-46.13.4	Torque/Force Reference Attributes.....	457
5-46.13.5	Current Loop Attributes.....	459
5-46.13.6	Frequency Control Attributes .....	462
5-46.13.7	Drive Output Attributes .....	463
5-46.14	Stopping & Braking Attributes .....	464

5-46.14.1	Semantics .....	466
5-46.14.2	Attribute #610 – Stopping Mode .....	466
5-46.14.3	Attribute #613 – Resistive Brake Contact Delay .....	467
5-46.14.4	Attribute #615 – Mechanical Brake Release Delay .....	467
5-46.14.5	Attribute #616 – Mechanical Brake Engage Delay .....	468
5-46.15	DC Bus Control Attributes.....	468
5-46.16	Power and Thermal Management Attributes .....	470
5-46.17	Axis Status Attributes .....	472
5-46.17.1	Semantics .....	473
5-46.17.2	Attribute #651 – Axis Status.....	473
5-46.17.3	Attribute #653 – Axis I/O Status .....	474
5-46.17.4	Attribute #94 – Status Data Set.....	475
5-46.18	Exception, Fault, and Alarm Status Attributes.....	475
5-46.18.1	Semantics .....	479
5-46.18.2	Standard Exception Table.....	479
5-46.19	Exception Limit Attributes.....	482
5-46.20	Exception Action Configuration Attribute.....	484
5-46.20.1	Semantics .....	484
5-46.20.2	Attribute #667-668 – Exception Action.....	484
5-46.21	Initialization Fault Status Attributes .....	485
5-46.21.1	Semantics .....	487
5-46.21.2	Standard Initialization Fault Table.....	487
5-46.22	Start Inhibit Status Attributes.....	488
5-46.22.1	Semantics .....	489
5-46.22.2	Standard Start Inhibit Table.....	489
5-46.23	Axis Statistical Attributes .....	489
5-46.24	Axis Info Attributes .....	489
5-46.25	General Purpose I/O Attributes.....	490
5-46.26	Local Mode Attributes .....	490
5-46.27	Common Services .....	491
5-46.28	Service Specific Data .....	492
5-46.28.1	GroupSync Service .....	492
5-46.29	Object Specific Services .....	493
5-46.30	Service Specific Data .....	493
5-46.30.1	Get Axis Attribute List .....	494
5-46.30.2	Set Axis Attribute List .....	497
5-46.30.3	Set Cyclic Write List.....	499
5-46.30.4	Set Cyclic Read List .....	500
5-46.30.5	Run Motor Test.....	501
5-46.30.6	Get Motor Test Data .....	501
5-46.30.7	Run Inertia Test .....	502
5-46.30.8	Get Inertia Test Data.....	502
5-46.30.9	Run Hookup Test.....	503
5-46.30.10	Get Hookup Test Data .....	503
5-46.31	Behavior.....	504
5-46.31.1	State Model.....	504
5-46.31.2	State Control .....	505
5-46.32	State Behavior .....	508
5-46.32.1	Off State .....	508
5-46.32.2	Self Test State .....	508
5-46.32.3	Initializing State.....	509
5-46.32.4	Pre-Charge State .....	509
5-46.32.5	Stopped State .....	509
5-46.32.6	Starting State.....	509
5-46.32.7	Running State.....	509

5-46.32.8	Testing State .....	510
5-46.32.9	Start Inhibited State .....	510
5-46.32.10	Stopping State.....	510
5-46.32.11	Aborting State.....	510
5-46.32.12	Major Faulted State.....	510
5-46.32.13	Shutdown State .....	510
5-46.33	Fault and Alarm Behavior.....	511
5-46.33.1	Exceptions.....	511
5-46.33.2	Exception Actions.....	511
5-46.33.3	Alarms.....	511
5-46.33.4	Major Faults.....	511
5-46.33.5	Minor Faults.....	512
5-46.33.6	Initialization Faults .....	512
5-46.33.7	Fault Codes .....	512
5-46.34	Start Inhibit Behavior.....	512
5-46.35	Visualization Behavior.....	512
5-46.35.1	Module Status LED .....	513
5-46.35.2	Axis Status LED .....	513
5-46.35.3	Alphanumeric Display .....	514
5-46.35.4	Seven-Segment Display .....	514
5-46.35.5	Multi-Character Alphanumeric Display.....	515
5-46.35.6	Multi-Axis Device Visualization .....	516
5-46.36	Command Generation Behavior.....	517
5-46.36.1	Command Data Sources.....	517
5-46.36.2	Command Fine Interpolation .....	517
5-46.36.3	Feed-Forward Signal Selection.....	520
5-46.36.4	Velocity Limiter.....	520
5-46.36.5	Skip Bands .....	520
5-46.36.6	Acceleration Limiter .....	520
5-46.37	Feedback Interface Behavior .....	521
5-46.37.1	Feedback Sources .....	521
5-46.37.2	Feedback Accumulator .....	522
5-46.37.3	Commutation Unwind and Offset.....	522
5-46.37.4	Feedback Filtering .....	522
5-46.38	Event Capture Behavior .....	524
5-46.38.1	Event Input Sources .....	524
5-46.38.2	Event Latches.....	524
5-46.38.3	Event Time Stamps .....	525
5-46.39	Control Mode Behavior .....	526
5-46.40	No Control (Feedback Only) Mode .....	526
5-46.41	Position Control Mode .....	527
5-46.42	Closed Loop Position Control.....	527
5-46.42.1	Position Feedback Configuration.....	527
5-46.42.2	Position PI Gains .....	527
5-46.42.3	Velocity Feed-Forward .....	527
5-46.43	Velocity Control Mode .....	528
5-46.44	Closed Loop Velocity Control .....	529
5-46.44.1	Velocity Feedback Configuration .....	529
5-46.44.2	Velocity Error Filter.....	529
5-46.44.3	Velocity PI Gains.....	529
5-46.44.4	Velocity Droop .....	530
5-46.44.5	Acceleration Feed-Forward .....	530
5-46.45	Open Loop Frequency Control .....	531
5-46.45.1	Basic Volts/Hertz Operation .....	531
5-46.45.2	Slip Compensation .....	532

5-46.45.3	Velocity Droop .....	532
5-46.46	Acceleration Control Mode.....	533
5-46.47	Torque Control Mode .....	533
5-46.47.1	Torque Input Sources.....	533
5-46.47.2	Torque Scaling.....	533
5-46.47.3	Low Pass Filter .....	534
5-46.47.4	Notch Filter.....	534
5-46.47.5	Torque Limiter.....	534
5-46.48	Current Control Mode.....	535
5-46.48.1	Current Vector Limiter .....	535
5-46.48.2	Voltage Output.....	535
5-46.48.3	Current Feedback.....	536
5-46.48.4	Motor Commutation .....	536
5-46.49	Definition of Motion Terms.....	537
5-47	Time Sync Object .....	539
5-47.1	CIP Sync Overview.....	539
5-47.2	Definitions.....	539
5-47.3	Overview of the Precision Time Protocol (PTP) .....	540
5-47.4	CIP Sync System Time Definition .....	541
5-47.5	CIP Sync Implementation .....	541
5-47.5.1	PTP Clock Implementation .....	542
5-47.5.2	PTP Protocol Stack Implementation .....	542
5-47.6	CIP Sync Clock Model .....	542
5-47.7	CIP Sync System Time Compensation .....	544
5-47.7.1	Step Compensation Algorithms .....	544
5-47.8	CIP Sync Group Startup Sequence .....	546
5-47.9	CIP Sync Quality of Service (QoS) .....	547
5-47.10	Time Sync Status Visualization .....	548
5-47.11	Time Sync Object Overview.....	548
5-47.12	Class Attributes .....	548
5-47.13	Instance Attributes .....	549
5-47.13.1	Semantics .....	550
5-47.14	Common Services .....	554
5-47.15	Object-Specific Services .....	554
5-47.15.1	Initialize Service .....	555
5-47.15.2	ManagementMessage Service.....	555
5-48	Connection Configuration Object .....	556
5-48.1	Class Attributes .....	556
5-48.1.1	Semantics .....	558
5-48.2	Instance Attributes .....	559
5-48.2.1	I/O Mapping Formats.....	562
5-48.2.2	Semantics .....	563
5-48.3	Connection Configuration Object Change Control.....	567
5-48.4	Common Services .....	567
5-48.4.1	Get_Attributes_All (Service Code 0x01).....	567
5-48.4.2	Set_Attributes_All (Service Code 0x02) .....	570
5-48.4.3	Create (Service Code 0x08) .....	571
5-48.4.4	Delete (Service Code 0x09) .....	572
5-48.4.5	Restore (Service Code 0x15) .....	572
5-48.4.6	Object-specific Services.....	573
5-48.4.7	Kick_Timer (Service Code 0x4B) .....	573
5-48.4.8	Open_Connection (Service Code 0x4C).....	573
5-48.4.9	Close_Connection (Service Code 0x4D) .....	574
5-48.4.10	Stop_Connection (Service Code 0x4E) .....	574
5-48.4.11	Change_Start (Service Code 0x4F) .....	574

5-48.4.12	Get_Status (Service Code 0x50).....	574
5-48.4.13	Change_Complete (Service Code 0x51).....	575
5-48.4.14	Audit_Changes (Service Code 0x52) .....	576
5-48.5	Behavior.....	577

## 5-1 Object Specifications

Using the format previously defined, the objects listed in Table 5-1.1 are specified in this object library. You can also use this table as a quick reference to objects and their corresponding class codes.

**Table 5-1.1 Object Specifications in the CIP Object Library**

Class Code	For information about this Object:	Go to page:
01hex	Identity Object	5-23
02hex	Message Router Object	5-37
03hex	DeviceNet Object	5-42
04hex	Assembly Object	5-43
05hex	Connection Object	5-50
06hex	Connection Manager Object	5-51
07hex	Register Object	5-52
08hex	Discrete Input Point Object	5-54
09hex	Discrete Output Point Object	5-61
0Ahex	Analog Input Point Object	5-71
0Bhex	Analog Output Point Object	5-79
0Ehex	Presence Sensing Object	5-90
0Fhex	Parameter Object	5-94
10hex	Parameter Group Object	5-108
12hex	Group Object	5-113
1Dhex	Discrete Input Group Object	5-118
1Ehex	Discrete Output Group Object	5-123
1Fhex	Discrete Group Object	5-129
20hex	Analog Input Group Object	5-133
21hex	Analog Output Group Object	5-139
22hex	Analog Group Object	5-146
23hex	Position Sensor Object	5-151
24hex	Position Controller Supervisor Object	5-171
25hex	Position Controller Object	5-179
26hex	Block Sequencer Object	5-189
27hex	Command Block Object	5-191
28hex	Motor Data Object	5-197
29hex	Control Supervisor Object	5-203
2Ahex	AC/DC Drive Object	5-216
2Bhex	Acknowledge Handler Object	5-226
2Chex	Overload Object	5-237
2Dhex	Softstart Object	5-240
2Ehex	Selection Object	5-245
30hex	S-Device Supervisor Object	5-259
31hex	S-Analog Sensor Object	5-277

Class Code	For information about this Object:	Go to page:
32hex	S-Analog Actuator Object	5-310
33hex	S-Single Stage Controller Object	5-318
34hex	S-Gas Calibration Object	5-338
35hex	Trip Point Object	5-344
N/A	Drive Data Object	5-350
37hex	File Object	5-351
38hex	S-Partial Pressure Object	5-368
39hex	Safety Supervisor Object	See Chapter 5 of Volume 5, CIP Safety
3Ahex	Safety Validator Object	See Chapter 5 of Volume 5, CIP Safety
3Bhex	Safety Discrete Output Point Object	See Chapter 5 of Volume 5, CIP Safety
3Chex	Safety Discrete Output Group Object	See Chapter 5 of Volume 5, CIP Safety
3Dhex	Safety Discrete Input Point Object	See Chapter 5 of Volume 5, CIP Safety
3Ehex	Safety Discrete Input Group Object	See Chapter 5 of Volume 5, CIP Safety
3Fhex	Safety Dual Channel Output Object	See Chapter 5 of Volume 5, CIP Safety
40hex	S-Sensor Calibration Object	5-379
41hex	Event Log Object	5-385
42 hex	Motion Axis Object	5-403
43 hex	Time Sync Object	5-539
44 hex	Modbus Object	See Chapter 5 of Volume 7, Integration of Modbus Devices into the CIP Architecture
F0hex	ControlNet Object	See Chapter 5 of Volume 4, ControlNet Adaptation of CIP
F1hex	ControlNet Keeper Object	See Chapter 5 of Volume 4, ControlNet Adaptation of CIP
F2hex	ControlNet Scheduling Object	See Chapter 5 of Volume 4, ControlNet Adaptation of CIP
F3hex	Connection Configuration Object	5-556
F4hex	Port Object	See Volume 1, Chapter 3
F5hex	TCP/IP Interface Object	See Chapter 5 of Volume 2, EtherNet/IP Adaptation of CIP
F6hex	Ethernet Link Object	See Chapter 5 of Volume 2, EtherNet/IP Adaptation of CIP
F7hex	CompoNet Link	See Chapter 5 of Volume 6, CompoNet Adaptation of CIP
F8hex	CompoNet Repeater	See Chapter 5 of Volume 6, CompoNet Adaptation of CIP

## 5-2 Identity Object

### Class Code: 01 Hex

This object provides identification of and general information about the device. The Identity Object shall be present in all CIP products.

If autonomous components of a device exist, use multiple instances of the Identity Object.

### 5-2.1 Class Attributes

**Table 5-2.1 Identity Object Class Attributes**

Number	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1	This class attribute is optional and is described in Chapter 4 of this specification.					
2	Conditional <sup>1</sup>	Get	Max Instance	UINT	Maximum instance number of an object currently created in this class level of the device.	The largest instance number of a created object at this class hierarchy level.
3 thru 7	These class attributes are optional and are described in Chapter 4 of this specification.					
1	If the possibility exists that multiple subcomponents will be identified using multiple instances of the Identity Object, then this attribute is required to be supported. The numerically lowest available integer shall be assigned as the Instance Identifier of a newly created Identity Object Instance.					

### 5-2.2 Instance Attributes

**Table 5-2.2 Identity Object Instance Attributes**

Attr ID	Need in Implem	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
1	Required	Get		Vendor ID	UINT	Identification of each vendor by number	See “Semantics” section
2	Required	Get		Device Type	UINT	Indication of general type of product	See “Semantics” section
3	Required	Get		Product Code	UINT	Identification of a particular product of an individual vendor	See “Semantics” section
4	Required	Get		Revision	STRUCT of:	Revision of the item the Identity Object represents	
				Major Revision	USINT		See “Semantics” section
				Minor Revision	USINT		See “Semantics” section
5	Required	Get		Status	WORD	Summary status of device	See “Semantics” section
6	Required	Get		Serial Number	UDINT	Serial number of device	See “Semantics” section
7	Required	Get		Product Name	SHORT_STRING	Human readable identification	See “Semantics” section

**Identity Object, Class Code: 01<sub>Hex</sub>**

Attr ID	Need in Implem	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
8	Optional	Get		State	USINT	Present state of the device as represented by the state transition diagram	0 = Nonexistent 1 = Device Self Testing 2 = Standby 3 = Operational 4 = Major Recoverable Fault 5 = Major Unrecoverable Fault 6 – 254 = Reserved 255 = Default for Get_Attributes_All service
9	Optional	Get		Configuration Consistency Value	UINT	Contents identify configuration of device	See “Semantics” section
10	Optional	Get/Set		Heartbeat Interval	USINT	The nominal interval between heartbeat messages in seconds.	The default value is 0. Zero disables transmission of the heartbeat message.
11	Optional	Set	NV	Active Language	STRUCT of	Currently active language for the device.	Based on ISO 639-2/T. See Appendix C-4, STRINGI data type, language1, language2, and language3 fields. Also see “Semantics” section.
					USINT	The language1 field from the STRINGI data type.	
					USINT	The language2 field from the STRINGI data type.	
					USINT	The language3 field from the STRINGI data type.	
12	Optional	Get	V	Supported Language List	ARRAY of STRUCT of	List of languages supported by character strings of data type STRINGI within the device.	See Appendix C-4, STRINGI data type, language1, language2, and language3 fields. The array index of the language is used within the International String Selection member protocol.
					USINT	The language1 field from the STRINGI data type.	
					USINT	The language2 field from the STRINGI data type.	
					USINT	The language3 field from the STRINGI data type.	
13	Optional	Get	NV	International Product Name	STRINGI	The names of the product in various languages	
14	Optional	Set		Semaphore	Struct of	Access Synchronization Semaphore	See “Semantics” section
				Client Electronic Key	UINT	Vendor Number	Default: All zero values
					UDINT	Client Serial Number	Timer valid range 100 thru 32767
				Semaphore Timer	ITIME	Millisecond Timer	
15	Optional	Set	NV	Assigned_Name	STRINGI	User assigned name.	See “Semantics” section
16	Optional	Set	NV	Assigned_Description	STRINGI	User assigned description.	See “Semantics” section
17	Optional	Set	NV	Geographic_Location	STRINGI	User assigned location.	See “Semantics” section

**Identity Object, Class Code: 01<sub>Hex</sub>**

Attr ID	Need in Implem	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
18	Conditional <sup>1</sup>			Modbus Identity Info		See Volume 7, Integration of Modbus Devices into the CIP Architecture (Chapter 5)	

1 This attribute is not allowed if the device is not a Modbus device. If the device is a Modbus device, see Volume 7.

The following Instance Attributes are used to identify with certainty the appropriate device targeted by a connection originator:

- Vendor ID
- Device Type
- Product Code
- Revision

This collection of attributes, when kept by the connection originator, is often referred to as a device's "electronic key".

### 5-2.2.1 Semantics

#### 5-2.2.1.1 Vendor ID

Vendor IDs are managed by the Open DeviceNet Vendor Association, Inc. (ODVA) and ControlNet International (CI). The value zero is not valid.

#### 5-2.2.1.2 Device Type

The list of device types is managed by ODVA and CI. It is used to identify the device profile that a particular product is using. Device profiles define minimum requirements a device must implement as well as common options.

A listing of the presently defined Device Types can be found in Chapter 6-1.

#### 5-2.2.1.3 Product Code

The vendor assigned Product Code identifies a particular product within a device type. Each vendor assigns this code to each of its products. The Product Code typically maps to one or more catalog/model numbers. Products shall have different codes if their configuration and/or runtime options are different. Such devices present a different logical view to the network. On the other hand for example, two products that are the same except for their color or mounting feet are the same logically and may share the same product code.

The value zero is not valid.

#### 5-2.2.1.4 Revision

The Revision attribute, which consists of Major and Minor Revisions, identifies the *Revision* of the item the Identity Object is representing.

The value zero is not valid for either the Major and Minor Revision fields.

The **Major** and **Minor** Revision are typically displayed as major.minor. Minor revisions shall be displayed as three digits with leading zeros as necessary. The Major Revision attribute is limited to 7 bits. The eighth bit is reserved by CIP and must have a default value of zero.

The major revision should be incremented by the vendor when there is a significant change to the 'fit, form, or function' of the product. Any changes that affect the configuration choices available to the user (and therefore the Electronic Data Sheet) require incrementing the major revision.

**Identity Object, Class Code: 01<sub>Hex</sub>**

The minor revision is typically used to identify changes in a product that do not affect user configuration choices. For example, bug fixes, hardware component change, labeling change, etc. Changes in minor revision are not used by a configuration tool to match a device with an Electronic Data Sheet.

**5-2.2.1.5 Status**

This attribute represents the current status of the entire device. Its value changes as the state of the device changes. The Status attribute is a WORD, with the following bit definitions:

**Table 5-2.3 Bit Definitions for Status Instance Attribute of Identity Object**

<b>Bit (s)</b>	<b>Called</b>	<b>Definition</b>
0	Owned	TRUE indicates the device (or an object within the device) has an owner. Within the Master/Slave paradigm the setting of this bit means that the Predefined Master/Slave Connection Set has been allocated to a master. Outside the Master/Slave paradigm the meaning of this bit is TBD.
1		Reserved, shall be 0
2	Configured	TRUE indicates the application of the device has been configured to do something different than the “out-of-box” default. This shall not include configuration of the communications.
3		Reserved, shall be 0
4 – 7	Extended Device Status	Vendor-specific or as defined by table below. The EDS shall indicate if the device follows the public definition for these bits using the DeviceStatusAssembly keyword in the [Device] section of the EDS. If these bits are vendor specific then they shall be enumerated in the EDS using the Assembly and Parameter sections.
8	Minor Recoverable Fault	TRUE indicates the device detected a problem with itself, which is thought to be recoverable. The problem does not cause the device to go into one of the faulted states. See Behavior section.
9	Minor Unrecoverable Fault	TRUE indicates the device detected a problem with itself, which is thought to be unrecoverable. The problem does not cause the device to go into one of the faulted states. See Behavior section.
10	Major Recoverable Fault	TRUE indicates the device detected a problem with itself, which caused the device to go into the “Major Recoverable Fault” state. See Behavior section.
11	Major Unrecoverable Fault	TRUE indicates the device detected a problem with itself, which caused the device to go into the “Major Unrecoverable Fault” state. See Behavior section.
12 - 15		Reserved, shall be 0

**Table 5-2.4 Bit Definitions for Extended Device Status Field**

<b>Bits 4 - 7:</b>	<b>Extended Device Status Description</b>
0 0 0 0	Self-Testing or Unknown
0 0 0 1	Firmware Update in Progress
0 0 1 0	At least one faulted I/O connection
0 0 1 1	No I/O connections established
0 1 0 0	Non-Volatile Configuration bad
0 1 0 1	Major Fault – either bit 10 or bit 11 is true (1)
0 1 1 0	At least one I/O connection in run mode
0 1 1 1	At least one I/O connection established, all in idle mode
1 0 0 0	Reserved, shall be 0
1 0 0 1	
1 0 1 0 thru 1 1 1 1	Vendor/Product specific

The values of the following *Status* bits indicate the state of the device:

**Identity Object, Class Code: 01<sub>Hex</sub>**

- Bit 8: Minor Recoverable Fault
- Bit 9: Minor Unrecoverable Fault
- Bit 10: Major Recoverable Fault
- Bit 11: Major Unrecoverable Fault

Note that the events that constitute a fault (recoverable or unrecoverable) are to be determined by the product developer. The following examples should help to define the various types of faults:

- Minor Recoverable Fault - an analog input device is sensing an input that exceeds the configured maximum input value.
- Minor Unrecoverable Fault - the device's battery backed RAM requires a battery replacement. The device will continue to function properly until the first time power is cycled.
- Major Recoverable Fault - the device's configuration is incorrect or incomplete.
- Major Unrecoverable Fault - the device failed its ROM checksum process.

**Table 5-2.5 Minor/Major Bit Definitions**

	<b>Recoverable</b>	<b>Non-recoverable</b>
<b>Minor (no state change)</b>	Bit 8	Bit 9
<b>Major (state changes)</b>	Bit 10	Bit 11

The event that sets the bit also causes a state change. See the State Transition Diagram in Figure 5-2.1.

**5-2.2.1.6 Serial Number**

This attribute is a number used in conjunction with the Vendor ID to form a unique identifier for each device on any CIP network. Each vendor is responsible for guaranteeing the uniqueness of the serial number across all of its devices.

**5-2.2.1.7 Product Name**

This text string should represent a short description of the product/product family represented by the product code in attribute 3. The same product code may have a variety of product name strings. The maximum number of characters in this string is 32.

**5-2.2.1.8 State**

This attribute is an indication of the present state of the device. Note that the nature of a Major Unrecoverable Fault could be such that it may not be accurately reflected by the State attribute.

**5-2.2.1.9 Configuration Consistency Value**

A product may automatically modify the *Configuration Consistency Value* whenever any non-volatile attribute is altered. A client node may, or may not, compare this value to a value within its own memory prior to system operation. The client node's behavior, upon detection of a mismatch, is vendor specific. The *Configuration Consistency Value* may be a CRC, incrementing count or any other mechanism. The only requirement is that if the configuration changes, the *Configuration Consistency Value* shall be different to reflect the change.

**5-2.2.1.10 Heartbeat Interval**

This attribute sets the nominal interval between production of optional heartbeat messages.

### 5-2.2.1.11 Active Language

If this attribute is supported, all character strings within the device of data type STRING or SHORT\_STRING shall follow this setting. Only languages, which are supported by the ISO-8859-1 character set, can be represented in these strings; if the Active Language is set to a language, which cannot be represented in ISO-8859-1, the device shall return the string in the default language. Any other object attributes, (class or instance level) which set the language for strings of these data types, shall not be supported (i.e. the class level Native Language attribute of the Parameter and Parameter Group objects).

### 5-2.2.1.12 Semaphore

This optional attribute provides a semaphore for client access synchronization to the entire device. This is a volatile attribute whose value after a reset or power-up is always zero. This attribute consists of two components, an electronic key of the client process and a semaphore timer.

The Client Electronic Key is composed of the client's CIP vendor number and the client's device serial number.

The Semaphore Timer is a millisecond timer, that when non-zero, counts down to zero. When the Semaphore Timer reaches the value zero, the Semaphore attribute is set to all zeros. The valid range of values for the timer portion of this attribute in a Set\_Attribute\_Single's service data is 100 milliseconds through 32767 milliseconds.

The Semaphore attribute can be read at any time. The value reported will be the current content of the Semaphore attribute, including the real-time (actual) timer value.

The Semaphore attribute's Semaphore Timer component operates at the base time resolution of the device. Therefore, if the time base of the device is greater than a 1 millisecond resolution, the time value accepted by the attribute shall be rounded up to the next timer increment appropriate for the device.

When a Set\_Attribute\_Single service is directed to this attribute, the attribute will be set to the service data if either of the following is true:

- The current value in the attribute is zero
- The Electronic Key portion of the service data matches the Electronic Key portion of the attribute data

The following error responses are defined for the Semaphore attribute:

Semaphore Attribute Error Responses		
Error Code	Additional Code	Error Response Interpretation
0x02	None	Resource Unavailable – The Client Electronic Key portion of the Semaphore attribute was non-zero and did not match the Client Electronic Key portion of the Set Attribute service data. This error response indicates that the Semaphore has been previously set by a different client process and is not available at the present time.
0x09	None	Invalid Attribute Value – The value specified for the timer portion of the attribute value was less than 100 milliseconds.
0x10	None	Device State Conflict – the device is currently in a state that does not allow the setting of the Semaphore attribute. This is a device specific behavior and shall be documented by the vendor.

**Identity Object, Class Code: 01<sub>Hex</sub>****5-2.2.1.13 Assigned\_Name**

This text string represents the user assigned name of the device.

**5-2.2.1.14 Assigned\_Description**

This text string represents a user assigned description of the device and may also describe its function.

**5-2.2.1.15 Geographic\_Location**

This text string represents the physical location of the device as provided by the user.

**5-2.3 Common Services**

The Identity Object provides the following Common Services:

**Table 5-2.6 Identity Object Common Services**

Service Code	Need in Implementation		Service Name	Description of Service
	Class	Instance		
0Ehex	Conditional <sup>1</sup>	Required	Get_Attribute_Single	Returns the contents of the specified attribute.
05hex	Optional	Required	Reset	Invokes the Reset service for the device.
01hex	Optional	Conditional	Get_Attributes_All	Returns a predefined listing of this objects attributes (See the Get_Attributes_All Response definition below)
10hex	n/a	Conditional	Set_Attribute_Single	Modifies an attribute (Required if Heartbeat interval is defined)
11hex	Optional	n/a	Find_Next_Object_Instance	Causes the specified Class to search for and return a list of instance IDs of existing instances of the Identity object.
18hex	Optional	Conditional <sup>2</sup>	Get_Member	Returns the content of a selected member of an attribute.

1 The Get\_Attribute\_Single service is REQUIRED if any Class attributes are implemented.

2 The Get\_Member service is required if any attributes with the International String (STRINGI) data type are implemented.

See Appendix A for definition of these services

**5-2.3.1 Reset Service**

When the Identity Object receives a Reset request, it:

- determines if it can provide the type of reset requested
- responds to the request
- attempts to perform the type of reset requested

The Reset common service has the following object-specific parameter:

**Table 5-2.7 Reset Service Parameter**

Name	Type	Description of Request Parameters	Semantics of Values
Type	USINT	Type of Reset	See Table below.

The parameter for the Reset common service has the following bit specifications:

**Identity Object, Class Code: 01<sub>Hex</sub>****Table 5-2.8 Reset Service Parameter Values**

<b>Value:</b>	<b>Type of Reset:</b>
0	Emulate as closely as possible cycling power on the item the <i>Identity Object</i> represents. This value is the default if this parameter is omitted.
1	Return as closely as possible to the factory default configuration, then emulate cycling power as closely as possible.
2	Return as closely as possible to the out-of-box configuration with the exception of communication link parameters and emulate cycling power as closely as possible. The communication link parameters that are to be preserved are defined by each network type. See the Reset service of the network specific link object(s) for complete information.
3 – 99	Reserved by CIP
100 – 199	Vendor specific
200 - 255	Reserved by CIP

**5-2.3.2 Get\_Attributes\_All Response**

At the **Class level**, the order of the attributes returned in the “Object/service specific reply data” portion of the Get\_Attributes\_All response is as follows:

**Table 5-2.9 Get\_Attributes\_All Response –Class Level**

<b>Byte</b>	<b>Bit 7</b>	<b>Bit 6</b>	<b>Bit 5</b>	<b>Bit 4</b>	<b>Bit 3</b>	<b>Bit 2</b>	<b>Bit 1</b>	<b>Bit 0</b>
0								Revision (low byte) Default = 1
1								Revision (high byte) Default = 0
2								Max Instance (low byte) Default = 1
3								Max Instance (high byte) Default = 0
4								Max ID Number of Class Attributes (low byte) Default = 0
5								Max ID Number of Class Attributes (high byte) Default = 0
6								Max ID Number of Instance Attributes (low byte) Default = 0
7								Max ID Number of Instance Attributes (high byte) Default = 0

**Important:** Insert default values for all unsupported attributes.

At the **Instance level**, the order of the attributes returned in the “Object/service specific reply data” portion of the Get\_Attributes\_All response is as follows:

**Table 5-2.10 Get\_Attributes\_All Response – Instance Level**

<b>Byte</b>	<b>Bit 7</b>	<b>Bit 6</b>	<b>Bit 5</b>	<b>Bit 4</b>	<b>Bit 3</b>	<b>Bit 2</b>	<b>Bit 1</b>	<b>Bit 0</b>
0								Vendor (low byte)
1								Vendor (high byte)
2								Device Type (low byte)
3								Device Type (high byte)
4								Product Code (low byte)
5								Product Code (high byte)
6								Major Revision
7								Minor Revision
8								Status (low byte)
9								Status (high byte)
10								Serial Number (low byte)

**Identity Object, Class Code: 01<sub>Hex</sub>**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
11								Serial Number
12								Serial Number
13								Serial Number (high byte)
14								Product Name length
15								Product Name (1st character) [See Note 1 below]
16								Product Name (2nd character)
n								Product Name (last character)
n+1								State, Default = 255
n+2								Configuration Consistency Value, Default = 0
n+4								Heartbeat Interval, Default = 0
n+5								Active Language, language1 field value
n+6								Active Language, language2 field value
n+7								Active Language, language3 field value
n+8								Supported Language List, Member 1, language1 field value
n+9								Supported Language List, Member 1, language2 field value
n+10								Supported Language List, Member 1, language3 field value
n+11								Supported Language List, Member 2, language1 field value
n+12								Supported Language List, Member 2, language1 field value
n+13								Supported Language List, Member 2, language1 field value
...								where m=number of members in Supported Language List and n=Product name Length+14 then j = n+8+((m-1)*3)
j								International Product Name (first octet) [See Note 1 below]
j+1								International Product Name (second octet)
...								
k								International Product Name (last octet)

**Note 1:** This is a variable length field.

**Important:** Insert default values for all unsupported attributes.

**Important:** Because the length of the name is not known before issuing the Get\_Attributes\_All service request, allow enough memory space to store a response up to 32 characters in length.

**Identity Object, Class Code: 01<sub>Hex</sub>****5-2.4 Object-specific Services**

The Identity Object provides no Object-specific services.

**5-2.5 Object-specific General and Extended Status Codes****Table 5-2.11 Object-specific General and Extended Status Codes**

<b>General Status Code (in hex)</b>	<b>8-bit Associated Extended Status Codes</b>	<b>Status Name</b>	<b>Description of Status</b>
00 - CF		General Codes	Defined in Appendix B
	00 - EE		Reserved Extended Status Codes
	F0 - FE	Vendor Specific	Vendor specific Extended Codes
	FF		Used with all General Codes when required and no other Extended Code is assigned
D0		Hardware Diagnostic	DeviceSelf Testing and Hardware Diagnostic Conditions
	00		Reserved
	01		Checksum (or CRC) error – Code space/ROM – Boot section
	02		Checksum (or CRC) error – Code space/ROM – Application section
	03		Checksum (or CRC) error – NV (flash/EEPROM) memory
	04		Invalid non-volatile (NV) memory – Configuration bad
	05		Invalid non-volatile (NV) memory – No configuration established
	06		RAM memory bad – The RAM memory in the device was determined to be experiencing inoperative cells
	07		ROM/Flash Memory bad
	08		Flash/EEPROM (NV) Memory Bad
	09		Interconnect wiring error / signal path problem
	0A		Power problem – Over current
	0B		Power problem – Over voltage
	0C		Power problem – Under voltage
	0D		Internal Sensor problem
	0E		System Clock Fault
	0F		Hardware configuration does not match NV configuration
	10		Watchdog Disabled/Idle
	11		Watchdog Timer Expired
	12		Device over temperature
	13		Ambient temperature outside of operating limits
	14 – EF	(reserved)	
	F0 – FE		Vendor specific Extended Codes
	FF		Used with all General Codes when required and no other Extended Code is assigned
D1		Device Status/States	Device Status Events and Conditions
	01		Power Applied
	02		Device Reset

**Identity Object, Class Code: 01<sub>Hex</sub>**

<b>General Status Code (in hex)</b>	<b>8-bit Associated Extended Status Codes</b>	<b>Status Name</b>	<b>Description of Status</b>
	03		Device Power Loss
	04		Activated
	05		Deactivated
	06		Enter Self-Test State
	07		Enter Standby State
	08		Enter Operational State
	09		Non-Specific Minor Recoverable Fault Detected
	0A		Non-Specific Minor Unrecoverable Fault Detected
	0B		Non-Specific Major Recoverable Fault Detected
	0C		Non-Specific Major Unrecoverable Fault Detected
	0D		Fault(s) corrected
	0E		CCV Changed
	0F		Heartbeat Interval Changed
	10 – EF	(reserved)	
	F0 - FE	Vendor Specific	Vendor Specific
	FF		Used with all General Codes when required and no other Extended Code is assigned
D2 – EF		Object Specific General Codes	Reserved by CIP – Not yet assigned
	00 - FF	Reserved	
F0 – FF		Vendor Specific General Codes	A vendor specific error has been encountered. The Additional Code Field of the Error Response defines the particular error encountered. Use of this General Error Code should only be performed when none of the Error Codes presented in this table or within an Object Class definition accurately reflect the error.
	00 - FF	Vendor Specific Extended Codes	All Extended Status Codes are available for association with each Vendor Specific General Code

**5-2.6 Behavior**

The behavior of the Identity Object is illustrated in the State Transition Diagram (STD) in Figure 5-2.1. This STD associates the state of the device with the status reported by the Status Attribute with the state of the Module Status LED.

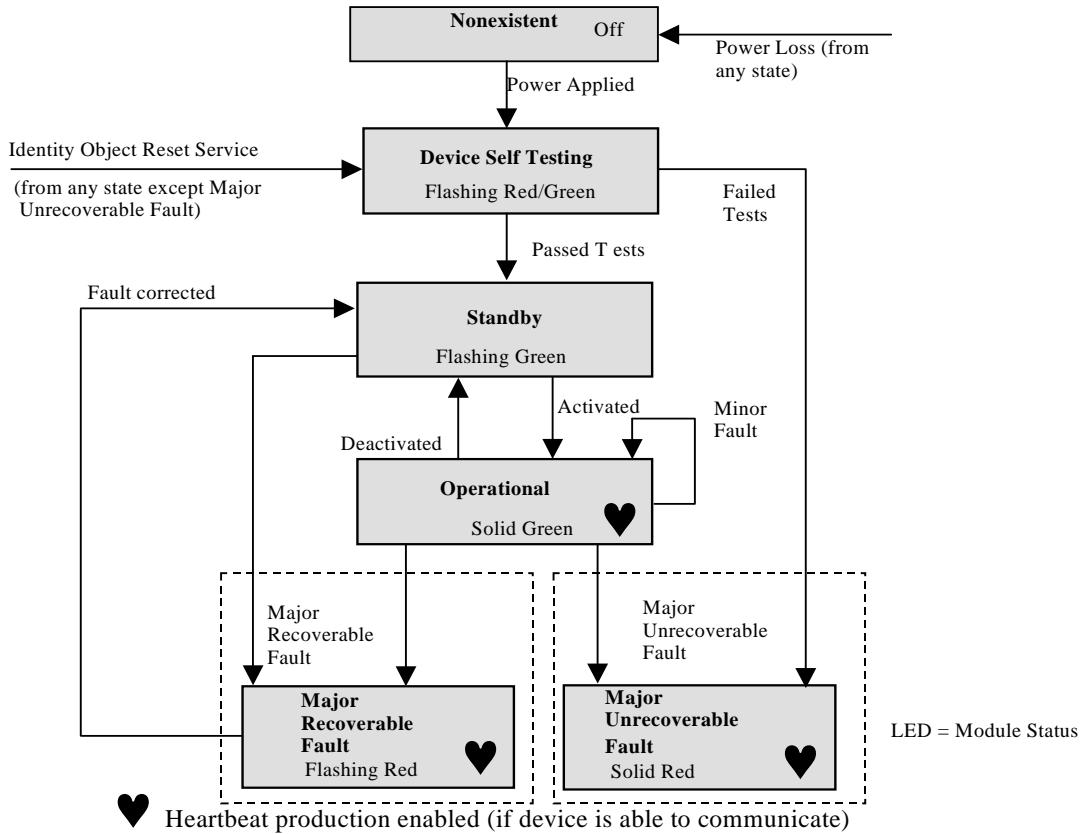
**Important:** A device may not be able to communicate in the Major Unrecoverable Fault state. Therefore, it might not be able to report a Major Unrecoverable Fault. It will not process a Reset service. The only exit from a Major Unrecoverable Fault is to cycle power.

The Identity object triggers production of heartbeat messages as defined by the underlying network when:

- the interval configured in the Heartbeat Interval Attribute has passed since the last heartbeat message.
- the Heartbeat message contents change, at a maximum rate of one “data changed” heartbeat message per second.

**Identity Object, Class Code: 01<sub>Hex</sub>**

The Heartbeat Interval value shall be saved as a Non-Volatile attribute. Heartbeat messages are only triggered after the device has successfully completed the network access state machine and is online. Not all networks support sending the Heartbeat message.

**Figure 5-2.1 State Transition Diagram for Identity Object**

The STD for the Identity object contains the following events:

- **Power Applied** - the device is powered up
- **Passed Tests** - the device has successfully passed all self tests
- **Activated** - the device's configuration is valid and the application for which the device was designed is now capable of executing (communications channels may or may not yet be established)
- **Deactivated** - the device's configuration is no longer valid and the application for which the device was designed is no longer capable of executing (communication channels may or may not still be established)
- **Minor fault** - a fault classified as either a minor unrecoverable fault or a minor recoverable fault has occurred
- **Major recoverable fault** - an event classified as Major Recoverable Fault has occurred
- **Major unrecoverable fault** - an event classified as a Major Unrecoverable Fault has occurred

Identity Object, Class Code: 01<sub>Hex</sub>

Table 5-2.12 State Event Matrix for Identity Object

Event	Nonexistent	Device Self Testing	Standby	Operational	Major Unrecoverable Fault	Major Recoverable Fault
Power Loss	Not Applicable	Transition to Nonexistent	Transition to Nonexistent	Transition to Nonexistent	Transition to Nonexistent	Transition to Nonexistent
Power Applied	Transition to Device Self Testing	Not Applicable	Not Applicable	Not Applicable	Not Applicable	Not Applicable
Failed Tests	Not Applicable	Transition to Major Unrecoverable Fault	Not Applicable	Not Applicable	Not Applicable	Not Applicable
Passed Tests	Not Applicable	Transition to Standby	Not Applicable	Not Applicable	Not Applicable	Not Applicable
Deactivated	Not Applicable	Ignore Event	Ignore Event	Transition to Standby	Ignore Event	Ignore Event
Activated	Not Applicable	Ignore Event	Transition to Operational	Ignore Event	Ignore Event	Ignore Event
Major Recoverable Fault	Not Applicable	Not Applicable	Transition to Major Recoverable Fault	Transition to Major Recoverable Fault	Ignore Event	Ignore Event
Major Unrecoverable Fault	Not Applicable	Not Applicable	Transition to Major Unrecoverable Fault	Transition to Major Unrecoverable Fault	Ignore Event	Ignore Event
Minor Recoverable Fault	Not Applicable	Ignore Event	Ignore Event	Ignore Event	Ignore Event	Ignore Event
Minor Unrecoverable Fault	Not Applicable	Ignore Event	Ignore Event	Ignore Event	Ignore Event	Ignore Event
Fault Corrected	Not Applicable	Not Applicable	Not Applicable	Not Applicable	Not Applicable	Transition to Standby
Reset	Not Applicable	Restart Self Tests	Transition to Device Self Testing	Transition to Device Self Testing	Ignore Event	Transition to Device Self Testing
Module Status LED	Off	Flashing Red/Green	Flashing Green	Solid Green	Solid Red	Flashing Red

The SEM for the Identity object contains the following states:

- **Nonexistent** - the device is without power
- **Device Self Testing** - the device is executing its self tests
- **Standby** - the device needs commissioning due to an incorrect or incomplete configuration
- **Operational** - the device is operating in a fashion that is normal for the device
- **Major Recoverable Fault** - the device has experienced a fault that is believed to be recoverable
- **Major Unrecoverable Fault** - the device has experienced a fault that is believed to be unrecoverable

**Identity Object, Class Code: 01<sub>Hex</sub>**

This page is intentionally left blank

## 5-3 Message Router Object

**Class Code: 02 hex**

The Message Router Object provides a messaging connection point through which a Client may address a service to any object class or instance residing in the physical device.

### 5-3.1 Class Attributes

**Table 5-3.1 Message Router Object Class Attributes**

Number	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1 thru 7	These class attributes are optional and are described in Chapter 4 of this specification.					

### 5-3.2 Instance Attributes

**Table 5-3.2 Message Router Object Instance Attributes**

Number	Need in implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1	Optional	Get	Object_list	STRUCT of	A list of supported objects	Structure with an array of object class codes supported by the device
			Number	UINT	Number of supported classes in the classes array	The number of class codes in the classes array
			Classes	ARRAY of UINT	List of supported class codes	The class codes supported by the device
2	Optional	Get	Number Available	UINT	Maximum number of connections supported	Count of the max number of connections supported
3	Optional	Get	Number active	UINT	Number of connections currently used by system components	Current count of the number of connections allocated to system communication
4	Optional	Get	Active Connections	ARRAY of: UINT	A list of the connection IDs of the currently active connections	Array of system connection IDs

### 5-3.3 Common Services

The Message Router Object provides the following Common Services:

**Table 5-3.3 Message Router Common Services**

Service Code	Need in Implementation		Service Name	Description of Service
	Class	Instance		
0Ehex	Conditional*	Conditional*	Get_Attribute_Single	Returns the contents of the specified attribute.
01hex	Optional	Optional	Get_Attributes_All	Returns the contents of all attributes

\*The Get\_Attribute\_Single service is REQUIRED if any attributes are implemented.

### 5-3.3.1 Get\_Attributes\_All Response

At the Class level, the order of the attributes returned in the “Object/service specific reply data” portion of the Get\_Attributes\_All response is as follows:

**Table 5-3.4 Get\_Attributes\_All Service Data – Class Level**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0								Revision (low byte) Default = 1
1								Revision (high byte) Default = 0
2								Optional Attribute List : number of attributes (low byte) Default = 0
3								Optional Attribute List : number of attributes (high byte) Default = 0
4								Optional Attribute List : optional attribute #1 (low byte)
5								Optional Attribute List : optional attribute #1 (high byte)
n								Optional Attribute List : optional attribute #m (low byte)
n+1								Optional Attribute List : optional attribute #m (high byte)
.								Optional Service List : number of services (low byte) Default = 0
.								Optional Service List : number of services (high byte) Default = 0
.								Optional Service List : optional service #1 (low byte)
.								Optional Service List : optional service #m (low byte)
.								Optional Service List : optional service #m (high byte)
.								Max ID Number of Class Attributes (low byte) Default = 0
.								Max ID Number of Class Attributes (high byte) Default = 0
.								Max ID Number of Instance Attributes (low byte) Default = 0
.								Max ID Number of Instance Attributes (high byte) Default = 0

**Important:** If the class attribute ”Optional Attribute List” is not supported, the default value of zero is to be inserted into the response byte array and **no** optional attribute numbers will follow.

**Important:** If the class attribute ”Optional Service List” is not supported, the default value of zero is to be inserted into the response byte array and **no** optional service numbers will follow.

**Important:** Insert default values for all unsupported attributes.

At the Instance level, the order of the attributes returned in the “Object/service specific reply data” portion of the Get\_Attributes\_All response is as follows:

**Message Router Object, Class Code: 02<sub>Hex</sub>****Table 5-3.5 Get\_Attributes\_All Service Data – Instance Level**

<b>Byte</b>	<b>Bit 7</b>	<b>Bit 6</b>	<b>Bit 5</b>	<b>Bit 4</b>	<b>Bit 3</b>	<b>Bit 2</b>	<b>Bit 1</b>	<b>Bit 0</b>
0								Object_list : Number (low byte) Default = 0
1								Object_list : Number (high byte) Default = 0
2								Object_list : Class #1 (low byte)
3								Object_list : Class #1 (high byte)
n								Object_list : Class #m (low byte)
n+1								Object_list : Class #m (high byte)
.								Number Available (low byte) Default = 0
.								Number Available (high byte) Default = 0
.								Number active (low byte) Default = 0
.								Number active (high byte) Default = 0
.								Active Connections #1 (low byte)
.								Active Connections #1 (high byte)
.								Active Connections #m (low byte)
.								Active Connections #m (high byte)

**Important:** Insert default values for all unsupported attributes.

**Important:** If the Instance attribute "Object\_list" is not supported, the default value of zero is to be inserted into the response byte array and **no** Object\_list class numbers will follow.

**Important:** If the Instance attribute "Number active" is not supported, the default value of zero is to be inserted into the response byte array and **no** Active Connection numbers will follow.

## 5-3.4 Object-Specific Services

The Message Router Object provides the following Object-specific services.

**Table 5-3.6 Object Specific Services**

<b>Service Code</b>	<b>Need in Implementation</b>		<b>Service Name</b>	<b>Description of Service</b>
	<b>Class</b>	<b>Instance</b>		
4B <sub>hex</sub>	Optional	n/a	SymbolicTranslation	Provides a translation from a Symbolic Segment EPATH encoding to the equivalent Logical Segment EPATH encoding, if it exists

### 5-3.4.1 Symbolic Translation

The SymbolicTranslation object specific service provides a mechanism for a device, that has implemented Symbolic Segment representations of internal EPATH addresses, to translate those Symbolic Addresses encodings into their equivalent Logical Segment EPATH encodings.

The service data of the SymbolicTranslation service shall be a single Symbolic Segment or ANSI Extended Symbol EPATH address. The response data shall be the equivalent Logical EPATH encoding.

**Message Router Object, Class Code: 02<sub>Hex</sub>****Table 5-3.7 SymbolicTranslation Request Service Data Field Parameters**

Name	Data Type	Description of Service Parameter
Symbolic Address	EPAUTH	A Symbolic Segment or ANSI Extended Symbol Segment to be translated

**Table 5-3.8 SymbolicTranslation Response Service Data Field Parameters**

Name	Data Type	Description of Service Parameter
Logical Address	EPAUTH	A Logical Segment and optional Data Segment encoding representing the internal equivalent of the requested Symbolic Address

**Table 5-3.9 SymbolicTranslation Service Error Response Values**

Error Code	Extended Code	Error Name	Description
0x20	0x00	Symbolic Path unknown	The Symbolic EPATH is not recognized by the processing node
	0x01	Symbolic Path destination not assigned	The Symbolic EPATH, although recognized, is not presently associated with a Logical EPATH equivalent.
	0x02	Symbolic Path segment error	The symbol identifier or the symbol segment syntax was not understood by the processing node.

### **5-3.5 Behavior**

The Message Router Object receives explicit messages and performs the following functions:

- interprets the Class Instance specified in a message
- routes a service to the specified object
- interprets services directed to it
- routes a response to the correct service source

#### **5-3.5.1 Service Request**

Interpretation of the Class Instance is performed on every service received by the Message Router.

Any Class Instance that cannot be interpreted by a device's implementation of a Message Router will report the Object\_Not\_Found error.

The service is then routed to a target object.

#### **5-3.5.2 Service Response**

All service responses are routed to the Explicit Messaging connection across which the service request was received.

## **5-4 DeviceNet Object**

**Class Code: 03 hex**

The DeviceNet Object provides the configuration and status of a DeviceNet port. Each DeviceNet product must support one (and only one) DeviceNet object per physical connection to the DeviceNet communication link.

See the Volume 3, DeviceNet Adaptation of CIP Common Specification for the definition of this object class.

## 5-5 Assembly Object

### Class Code: 04<sub>hex</sub>

The Assembly Object binds attributes of multiple objects, which allows data to or from each object to be sent or received over a single connection. Assembly objects can be used to bind input data or output data. The terms "input" and "output" are defined from the network's point of view. An input will produce data on the network and an output will consume data from the network.

**Table 5-5.1 Revision History**

Assembly Class Revision	Description
00	Pre-release definition
01	Initial release
02	1. Class specific Service Codes 4B <sub>hex</sub> and 4C <sub>hex</sub> obsoleted

Assembly objects instances can either be dynamic or static:

- **Dynamic:** assemblies with member lists created and managed by the user. The member list can be altered by adding or deleting members. Dynamic assemblies shall be assigned instance IDs in the vendor specific range.
- **Static:** assemblies with member lists defined by the device profile or by the manufacturer of the product. The Instance number, number of members, and member list are fixed. Static assemblies can usually be implemented entirely in ROM.

**Important:** Instances of the Assembly Object are divided into the following address ranges to provide for extensions to device profiles.

**Table 5-5.2 Assembly Instance ID Ranges**

Range	Meaning	Quantity
01 - 63 <sub>hex</sub>	Open (static assemblies defined in device profile)	99
64 <sub>hex</sub> - C7 <sub>hex</sub>	Vendor Specific static assemblies and dynamic assemblies	100
C8 <sub>hex</sub> - 2FF <sub>hex</sub>	Open (static assemblies defined in device profile)	568
300 <sub>hex</sub> - 4FF <sub>hex</sub>	Vendor Specific static assemblies and dynamic assemblies	512
500 <sub>hex</sub> - FFFF <sub>hex</sub>	Open (static assemblies defined in device profile)	1,047,296
100000 <sub>hex</sub> - FFFFFFFF <sub>hex</sub>	Reserved by CIP for future use	4,293,918,720

## 5-5.1 Class Attributes

**Table 5-5.3 Assembly Object Class Attributes**

Attribute ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1	Conditional*	Get	Revision	UINT	Revision of this object	The current value assigned to this attribute is two (02).
2 thru 7	These class attributes are optional and are described in Chapter 4 of this specification.					

\*This attribute is REQUIRED if Instance Attribute 2 is supported, otherwise this attribute is OPTIONAL.

## 5-5.2 Instance Attributes

**Table 5-5.4 Assembly Object Instance Attributes**

Attribute ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1	Conditional	Get	Number of Members in List	UINT		Required for Dynamic Assembly only
2	Conditional	Set for Dynamic/ Get for Static	Member List	ARRAY of STRUCT:	The member list is an array of CIP paths	Required for Dynamic Assembly only
			Member Data Description	UINT	Size of member data.	Size in bits
			Member Path Size	UINT	Size of Member Path (in bytes).	
			Member Path	Packed EPATH	See Appendix C for the format of this field.	
3	Required	Set	Data	ARRAY of BYTE		
4	Optional	Get	Size	UINT	Number of bytes in Attribute 3	

### 5-5.3 Common Services

The Assembly Object provides the following Common Services:

**Table 5-5.5 Assembly Object Common Services**

Service Code	Need in Implementation				Service Name	Description of Service		
	Static Assembly		Dynamic Assembly					
	Class	Instance	Class	Instance				
0Ehex	Conditional <sup>1</sup>	Required	Required	Required	Get_Attribute_Single	Returns the contents of the specified attribute.		
08hex	n/a	n/a	Required	n/a	Create	Instantiates an Assembly Object within a specified class. Response contains instance number. Dynamic assemblies shall be assigned instance IDs in the vendor specific range.		
10hex	n/a	Optional	n/a	Conditional <sup>2</sup>	Set_Attribute_Single	Modifies an attribute value.		
09hex	n/a	n/a	Optional <sup>3</sup>	Required	Delete	Deletes an Assembly Object and releases all associated resources.		
1Ahex	n/a	n/a	n/a	Conditional <sup>4</sup>	Insert_Member	Adds a member to the Assembly Member List.		
1Bhex	n/a	n/a	n/a	Conditional <sup>4</sup>	Remove_Member	Removes a member from the Assembly Member List.		
18hex	n/a	Optional	n/a	Optional	Get_Member	Returns a member from the Assembly Member List or Data attributes.		
19hex	n/a	Optional	n/a	Optional	Set_Member	Modifies a member of the Assembly Member List or Data attributes.		

1 Required if Max Instance Class Attribute is implemented or Instance Attribute 2 (Member List).

2 If you choose NOT to support the Set\_Attribute\_Single common service at the Instance level for a Dynamic Assembly, then your product shall support the Insert\_Member and Remove\_Member common services.

3 At the class level this service deletes all existing Assembly instances.

4 If you choose NOT to support the Insert\_Member and Remove\_Member common services at the Instance level for a Dynamic Assembly, then your product shall support the Set\_Attribute\_Single common service.

See Appendix A for definition of these services

## 5-5.4 Object-specific Services

The Assembly Object provides no Object-specific services. The following Object-specific service codes have been obsoleted:

**Table 5-5.6 Assembly Object Obsolete Object-Specific Services**

Obsolete Service Code	Need in Implementation				Service Name	Description of Service		
	Static Assembly		Dynamic Assembly					
	Class	Instance	Class	Instance				
4Bhex	n/a	n/a	n/a	n/a	Add_Member	Obsolete		
4Chex	n/a	n/a	n/a	n/a	Remove_Member	Obsolete		

## 5-5.5 Behavior

The behavior of the Assembly Object differs by the type of Assembly: *dynamic* or *static*. A Dynamic Assembly's member list is created and managed by the user of the device. The manager of the dynamic assembly must specify and maintain the member list.

A Static Assembly's member list is defined by the device manufacturer. It cannot be modified.

To provide for the ability to create and delete objects, as well as change member lists, Dynamic Assemblies support additional services which are not supported by Static Assemblies.

The following rules apply to the member lists of both static and dynamic assemblies.

When an empty path (Member Path Size = 0) is used in an assembly member list, the assembly inserts/discards the number of bits as specified in the Member Data Size field when producing/consuming. The assembly shall use a value of zero for all produced data which has been inserted. The use of an empty consumed path allows, for example, data destined for multiple nodes to be sent in a single message since each node can be configured to discard data in the message not intended for it.

The empty path shall be supported for all dynamic assemblies.

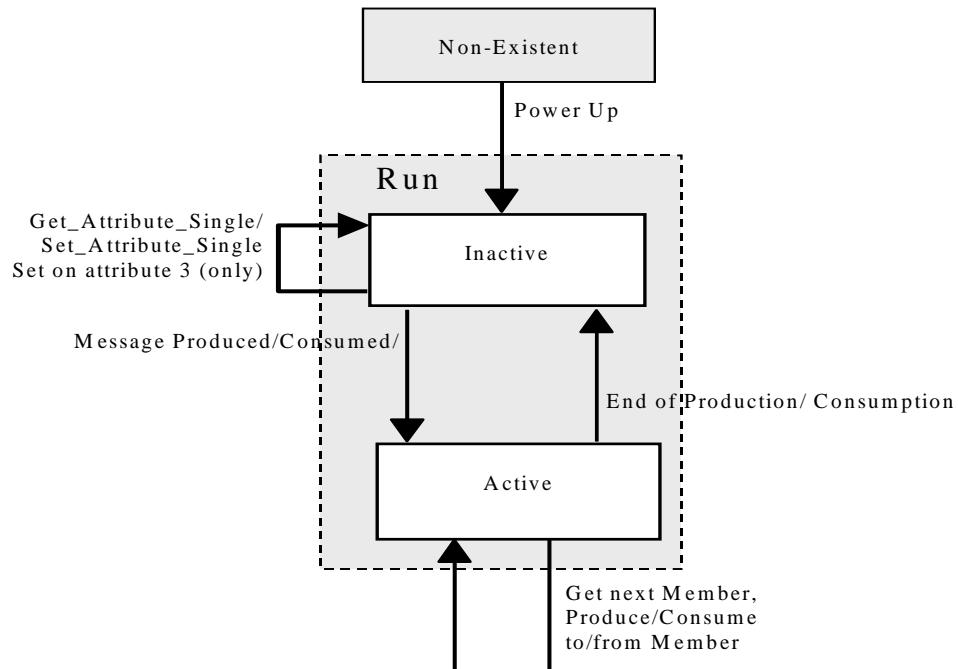
No checking is done by the Assembly Object at any time to verify that the size of the member data is correct for the given member path. It is the responsibility of the assembly member to properly handle too much or too little data. The assembly is required to deliver the configured number of bits to the member.

No padding is done by the Assembly Object to align data from each assembly member on a byte, word, or other boundary.

### 5-5.5.1 Static Assemblies

The following State Transition Diagram, State Event Matrix and Attribute Access Table illustrate the behavior of Static assemblies.

**Figure 5-5.7 Static Assembly State Transition Diagram**



**Table 5-5.8 Static Assembly State Event Matrix**

Event	Static Assembly Object State		
	Non-existent	Inactive	Active
Power Up	Transition to Inactive	Not applicable	Not applicable
Get_Attribute_Single	Error: Object does not exist. (General Error Code 16 <sub>hex</sub> )	Validate/service the request Return response	Validate/service the request Return response
Set_Attribute_Single	Error: Object does not exist. (General Error Code 16 <sub>hex</sub> )	Validate/service the request Return response	Error: Object State Conflict (General Error Code 0C <sub>hex</sub> )
Message produced/consumed	Error: Object does not exist. (General Error Code 16 <sub>hex</sub> )	Begin producing/consuming from/to each member in list Transition to Active	Error: Object State Conflict (General Error Code 0C <sub>hex</sub> )
End of production/consumption	Error: Object does not exist. (General Error Code 16 <sub>hex</sub> )	Error: Object State Conflict (General Error Code 0C <sub>hex</sub> )	Transition to Inactive

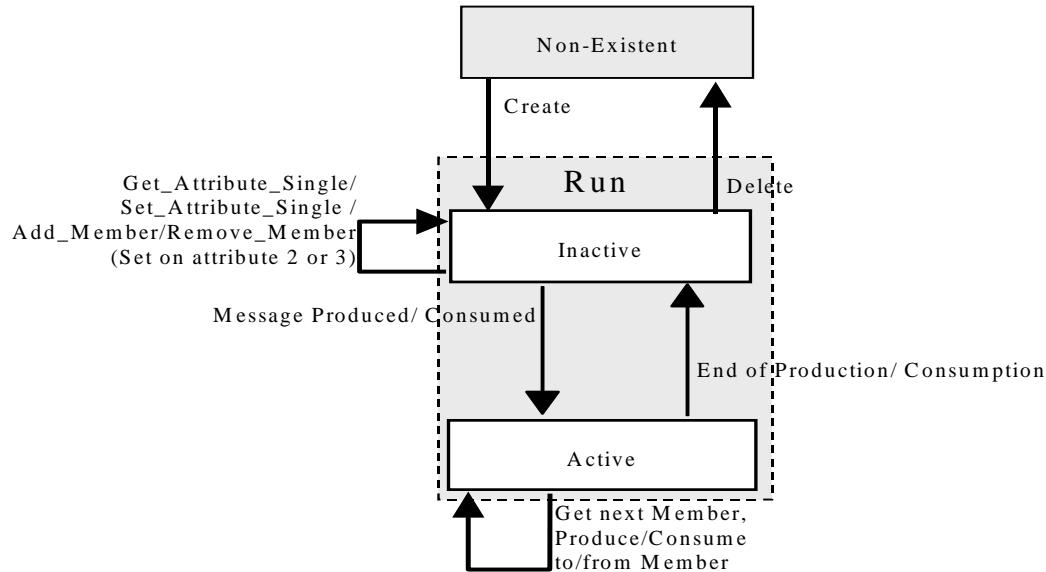
**Table 5-5.9 Static Assembly Object Attribute Access**

Attribute	Static Assembly Object State		
	Non-existent	Inactive	Active
Number_of_Members	Not available	Read Only	Read Only
Member_List	Not available	Read Only	Read Only
Data	Not available	Read/Write	Read Only

### 5-5.5.2 Dynamic Assemblies

The State Transition Diagram, State Event Matrix and Attribute Access Table below illustrate the behavior of Dynamic assemblies.

**Figure 5-5.10 Dynamic Assembly State Transition Diagram**



**Table 5-5.11 Dynamic Assembly State Event Matrix**

Event	Dynamic Assembly Object State		
	Non-Existent	Inactive	Active
Create	Class instantiates an Assembly Object. Transition to Inactive	Not applicable	Not applicable
Delete	Error: Object does not exist. (General Error Code 0x16)	Release all associated resources. Transition to Non-Existent	Error: Object State Conflict (General Error Code 0x0C)
Get_Attribute_Single	Error: Object does not exist. (General Error Code 0x16)	Validate/service the request. Return response	Validate/service the request. Return response
Set_Attribute_Single	Error: Object does not exist. (General Error Code 0x16)	Validate/service the request. Return response	Error: Object State Conflict (General Error Code 0x0C)
Insert_Member	Error: Object does not exist. (General Error Code 0x16)	Validate/service the request. Return response	Error: Object State Conflict (General Error Code 0x0C)
Remove_Member	Error: Object does not exist. (General Error Code 0x16)	Validate/service the request. Return response	Error: Object State Conflict (General Error Code 0x0C)
Message produced/ consumed	Error: Object does not exist. (General Error Code 0x16)	Begin producing/consuming from/to each member in list. Transition to Active.	Error: Object State Conflict (General Error Code 0x0C)
End of production/ consumption	Error: Object does not exist. (General Error Code 0x16)	Error: Object State Conflict (General Error Code 0x0C)	Transition to Inactive

**Table 5-5.12 Dynamic Assembly Object Attribute Access**

Attribute	Dynamic Assembly Object State		
	Non-existent	Inactive	Active
Number_of_Members	Not available	Read Only	Read Only
Member_List <sup>1</sup>	Not available	Read/Write	Read Only
Data	Not available	Read/Write	Read Only

1 This attribute can be set by either the Insert\_Member service (one member at a time) or the Set\_Attribute\_Single service (all members at once).

### 5-5.3 Connection Points

Connection Points within the Assembly Object are identical to Instances. For example, Connection Point 4 of the Assembly Object is the same as Instance 4. Specifying an EPATH of “20 04 24 VV 30 03” is the same as “20 04 2C VV 30 03”.

## **5-6 Connection Object**

**Class Code: 05<sub>hex</sub>**

Use the Connection Object to manage the characteristics of a communication connection.

See Chapter 3 for the definition of this object class.

## **5-7 Connection Manager Object**

**Class Code: 06 hex**

Use this object for connection and connectionless communications, including establishing connections across multiple subnets.

See Chapter 3 for the definition of this object class.

## 5-8 Register Object

### Class Code: 07 hex

Use this object to address individual bits or a range of bits up to 64K bits of data.

Note that a Register object can operate as either an input register or an output register. The terms “input” and “output” are defined from the network’s point of view. An input will produce data on the network and an output will consume data from the network.

### 5-8.1 Class Attributes

**Table 5-8.1 Register Object Class Attributes**

Attribute ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1 thru 7	These class attributes are optional and are described in Chapter 4 of this specification.					

### 5-8.2 Instance Attributes

**Table 5-8.2 Register Object Class Attributes**

Attribute ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1	Required	Get	Bad Flag	BOOL		0=good 1=bad
2	Required	Set	Direction	BOOL	Direction of data transfer	0=Input Register, 1=Output Register
3	Required	Set	Size	UINT	Size of register data in bits	
4	Required	Set (Set is optional if Direction=1)	Data	ARRAY of BITS	Data to be transferred	*

\* All encoded bits are to be LSB aligned, as shown in this example.

7 . . . . . 0	15 . . . . . 8	23 . . . . . 16	29 . . . . . 24	30 bit array encoding example
---------------	----------------	-----------------	-----------------	-------------------------------

### 5-8.3 Common Services

The Register Object provides the following Common Services:

**Table 5-8.3 Register Object Common Services**

Service Code	Need in Implementation		Service Name	Description of Service
	Class	Instance		
0Ehex	Conditional*	Required	Get_Attribute_Single	Returns the contents of the specified attribute.
10hex	n/a	Optional	Set_Attribute_Single	Modifies an attribute value.

\*The Get\_Attribute\_Single service is REQUIRED if any attributes are implemented.

See Appendix A for definition of these services

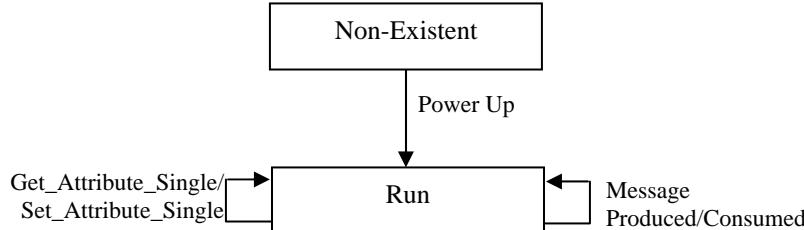
### 5-8.4 Object-specific Services

The Register Object provides no Object-specific services.

### 5-8.5 Behavior

The State Transition Diagram, State Event Matrix, and Attribute Access Table below illustrate the behavior of the Register Object.

**Figure 5-8.4 Register Object State Transition Diagram**



**Table 5-8.5 Register Object State Event Matrix**

Event	State	
	Non-existent	Run
Power up	Transition to Run	Not applicable.
Get_Attribute_Single	Error: Object does not exist (General Error Code 16 <sub>hex</sub> )	Validate/service the request. Return response.
Set_Attribute_Single	Error: Object does not exist (General Error Code 16 <sub>hex</sub> )	Validate/service the request. Return response.
Message produced/consumed	Error: Object does not exist (General Error Code 16 <sub>hex</sub> )	Apply/retrieve data.

## 5-9 Discrete Input Point Object

### Class Code: 08<sub>hex</sub>

The Discrete Input Point (DIP) Object models discrete inputs in a product. You can use this object in applications as simple as a toggle switch or as complex as a discrete I/O control module. Note that the term "input" is defined from the network's point of view. An input will produce data on the network.

The Discrete Input Point interface is to real input points such as a switch or screw terminal. The input is sampled and the data is stored in this object's VALUE attribute. A sample of the discrete input value is triggered via an external command (input change-of-state, cyclic data trigger, etc.)

### 5-9.1 Revision History

Since the initial release of this object class definition changes have been made that require a revision update of this object class. The table below represents the revision history:

**Table 5-9.1 Discrete Input Point Object Revision History**

Revision	Reason for object definition update
01	Initial Definition at First Release of Specification
02	IDLE state removed from this object's behavior

### 5-9.2 Class Attributes

**Table 5-9.2 Discrete Input Point Class Attributes**

Attribute ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1	Required	Get	Revision	UINT	Revision of this object	The current value assigned to this attribute is two (02).
2 thru 7	These class attributes are optional and are described in Chapter 4 of this specification.					

### 5-9.3 Instance Attributes

Table 5-9.3 Discrete Input Point Object Instance Attributes

Attribute ID	Need in Implem.	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
1	Optional	Get		Number of Attributes	USINT	Number supported in this product	
2	Optional	Get		Attribute List	ARRAY OF USINT	List of attributes supported in this product	
3	Required	Get		Value	BOOL	Input point value	0=off; 1=on
4	Optional	Get		Status	BOOL	Input point status	0=OK; 1=product specific alarm or status
5	Optional	Set		Off_On Delay	UINT	filter time for off to on transition 0 - 65,535 microseconds <sup>1</sup>	The default value is 0.
6	Optional	Set		On_Off Delay	UINT	filter time for on to off transition 0 - 65,535 microseconds <sup>2</sup>	The default value is 0.
7	Optional	Set <sup>3</sup>	NV	Off_On Cycles	UDINT	Total number of times the Value attribute transitioned from the Off to the On state	[default] = 0

- 1 The input must be on for the amount of filter time specified by the OFF\_ON DELAY attribute before the ON state is recorded in the VALUE attribute.
- 2 The input must be off for the amount of filter time specified by the ON\_OFF DELAY attribute before the OFF state is recorded in the VALUE attribute.
- 3 Applications including this attribute may restrict its access to Get Only; thus, allowing the application support of a factory initialized cycle life indication.

### 5-9.4 Common Services

The Discrete Input Point Object provides the following Common Services:

Table 5-9.4 Discrete Input Point Object Common Services

Service Code	Need in Implementation		Service Name	Description of Service
	Class	Instance		
0Ehex	Required	Required	Get_Attribute_Single	Returns the contents of the specified attribute.
01hex	Optional	Optional	Get_Attributes_All	Returns a predefined listing of this objects attributes (See the Get_Attributes_All Response definition below)
10hex	n/a	Optional	Set_Attribute_Single	Modifies an attribute value.
02hex	Optional	Optional	Set_Attributes_All	Modifies the value of a list of attributes (See the Set_Attributes_All Request definition below)

See Appendix A for definitions of these common services.

### 5-9.4.1 Get\_Attributes\_All Response

At the Class level, the order of the attributes returned in the “Object/service specific reply data” portion of the Get\_Attributes\_All response is as follows: (see Chapter 4 for a description of the Get\_Attributes\_All Service Data field)

**Table 5-9.5 Get\_Attributes\_All Response Data – Class Level**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0								Revision (low byte)
1								Revision (high byte) Default = 0
2								Max Instance (low byte) Default = 0
3								Max Instance (high byte) Default = 0
4								Max ID Number of Class Attributes (low byte) Default = 1
5								Max ID Number of Class Attributes (high byte) Default = 0
6								Max ID Number of Instance Attributes (low byte) Default = 3
7								Max ID Number of Instance Attributes (high byte) Default = 0

**Important:** Insert default values for all unsupported attributes.

At the **Instance level**, the order of the attributes returned in the “Object/service specific reply data” portion of the Get\_Attributes\_All response is as follows:

**Table 5-9.6 Get\_Attributes\_All Response Data – Instance Level**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0								Number of Attributes Default = 0
1								Attribute List (attribute #1)
n								Attribute List (attribute #m)
n+1	0	0	0	0	0	0	0	Value
n+2	0	0	0	0	0	0	0	Status Default = 0
.								Off_On Delay (low byte) Default = 0
.								Off_On Delay (high byte) Default = 0
.								On_Off Delay (low byte) Default = 0
.								On_Off Delay (high byte) Default = 0

**Important:** Insert default values for all unsupported attributes.

**Important:** If the instance attribute ”Number of Attributes” is not supported, the default value of zero is to be inserted in its place and **no** members of the ”Attribute List” attribute will follow.

### 5-9.4.2 Set\_Attributes\_All Request

No settable attributes currently exist at the Class level for the Discrete Input Point Object.

At the Instance level, the order of attributes passed in the Set\_Attributes\_All request is as follows:

**Table 5-9.7 Set\_Attributes\_All Request Service Data – Instance Level**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0								Off_On Delay (low byte)
1								Off_On Delay (high byte)
2								On_Off Delay (low byte)
3								On_Off Delay (high byte)

**Important:** The Set\_Attributes\_All service is to be supported only if **all** settable attributes shown above are implemented as settable.

### 5-9.5 Object-specific Services

The DIP Object provides no Object-specific services.

### 5-9.6 Behavior

The State Transition Diagram in Figure 5-9.6.1 provides a graphical description of the events and corresponding state transitions. A subset of the states and events may be supported in an application, but the behavior must be consistent.

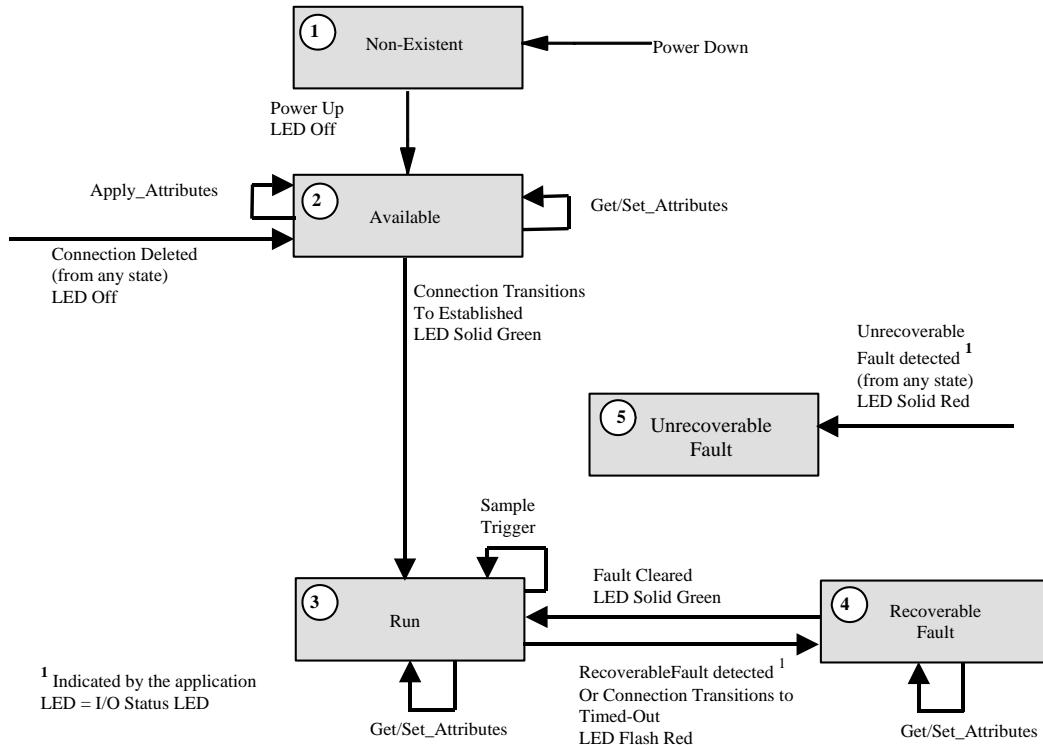
The State Event Matrix in Table 5-9.6.1. lists all pertinent events and the corresponding action to be taken while in each state.

**Important:** Events can occur simultaneously, but the *Fault* events have priority if they occur simultaneously with other events.

The following behavior is related to the *Off\_On Cycles* attribute. The value of *Off\_On Cycles* is incremented once for each time the input transitions from an ‘Off’ state to a ‘On’ state. The purpose is to assist in tracking the cycles the device that is reporting the input has encountered. Upon reaching a terminal count (maximum value) and experiencing another cycle to count, the *Off\_On Cycles* value will “wrap” to a value of zero (0) and continue to count cycles as described.

Discrete Input Point Object, Class Code: 08<sub>Hex</sub>

Figure 5-9.8 State Transition Diagram for Discrete Input Point Object



**Important:** Events can occur simultaneously, but the *Fault* events have priority if they occur simultaneously with other events.

The following SEM contains these states:

- **Non-Existent:** a module without power.
- **Available:** waiting for a connection, power-up discrete input point defaults are set.
- **Run:** DIP sensing data from its input and transmitting the data.
- **Recoverable Fault:** a recoverable fault has occurred.
- **Unrecoverable Fault:** an unrecoverable fault has occurred.

The SEM also contains these events:

Table 5-9.9 Discrete Input Point Object Event Definitions

This event	Is
Sample Trigger	a change of state, cyclic timer trigger, application trigger
Connection Deleted	I/O connection deleted.
Apply_Attributes	the Apply service of the I/O connection object the Discrete Input Point object is connected to. Note: the application is responsible for validating the connection object's attributes.
Fault Cleared	the application clearing a detected fault
Connection Transitions to Established	I/O connection transitions to Established.
Connection Transitions to Timed Out	I/O connection transitions to Timed-Out

Discrete Input Point Object, Class Code: 08<sub>Hex</sub>

The figure below is a conceptual illustration of the state machine for a typical input object (producing application). The events listed above are represented by the dotted line labeled “state change.”

Figure 5-9.10 State Machine Illustration for Typical Discrete Input Point Object

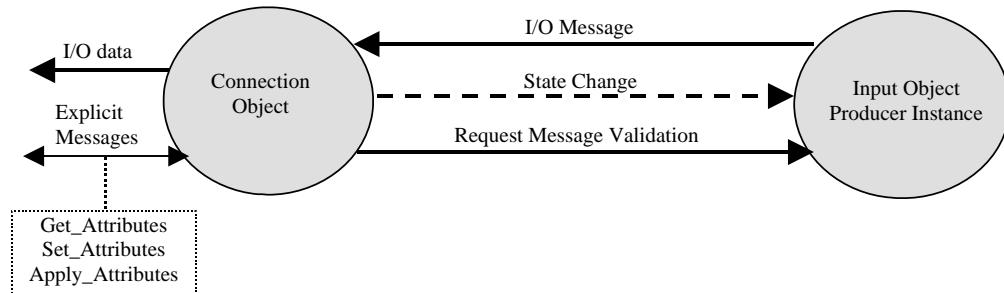


Table 5-9.11 State Event Matrix for the Discrete Input Point Object

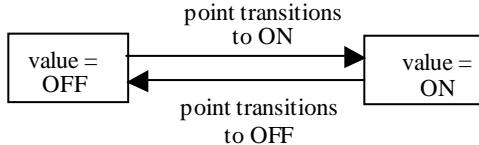
Event	State				
	Non-Existent	Available	Run	Recoverable Fault	Unrecoverable Fault
Sample Trigger	Not Applicable	Ignore event	Sample data, Send data	Ignore event	Ignore event
Apply Attributes	Not Applicable	Verify attributes, return result	Return error (Object State Conflict)	Return error (Object State Conflict)	Ignore event
Connection Deleted	Not Applicable	Ignore Event	Transition to Available	Transition to Available	Ignore event
Connection Transitions to Established	Not Applicable	Transition to Run	Ignore event	Ignore event	Ignore event
Connection Transitions to Timed Out state	Not Applicable	Ignore event	Transition to Recoverable Fault	Ignore event	Ignore event
Fault Cleared	Not Applicable	Not Applicable	Not Applicable	Transition to Run	Ignore event
Get_Attribute	Return Error (Object Does Not Exist)	Return value	Return value	Return value	Ignore event
Set_Attribute	Return Error (Object Does Not Exist)	Accept value	Accept value	Accept value	Ignore event
I/O Status LED	Off	Off	Solid Green	Flash Red	Solid Red

### 5-9.6.1 Attribute Access Rules

Except in the Non-Existent and Unrecoverable Fault states, all attributes are gettable or settable according to their access rules.

Because the only required Instance attribute is *Value*, the only required behavior of a Discrete Input Point is that it indicates a boolean value of OFF or ON. The optional attributes either provide more information about the discrete input point or alter the behavior of the input point.

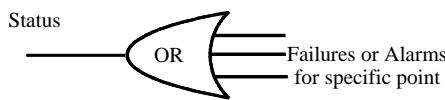
**Figure 5-9.12 Discrete Input Point Object Value Attribute Illustration**



### 5-9.6.1.1 Status Instance Attribute

The optional *Status* Instance attribute is simply a logical OR of all possible failure or alarm conditions for the point.

**Figure 5-9.13 Discrete Input Point Object Status Attribute Function**



## 5-10 Discrete Output Point Object

**Class Code: 09<sub>hex</sub>**

A Discrete Output Point (DOP) models discrete outputs in a product. You can use this object in applications such as discrete I/O control modules, relays, switches, etc. Note that the term “output” is defined from the network’s point of view. An output will consume data from the network.

The Discrete Output Point interface is to real output points such as a relay or LED. The output is read from this object’s VALUE attribute and applied to the output terminal (e.g. screw terminal).

### 5-10.1 Class Attributes

**Table 5-10.1 Discrete Output Point Object Class Attributes**

Attribute ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1 thru 7	These class attributes are optional and are described in Chapter 4 of this specification.					

### 5-10.2 Instance Attributes

**Table 5-10.2 Discrete Output Point Object Instance Attributes**

Attribute ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1	Optional	Get	Number of Attributes	USINT	Number of attributes supported in this product	
2	Optional	Get	Attribute List	ARRAY OF USINT	List of attributes supported in this product	
3	Required	Set	Value	BOOL	Output point value	0=off; 1=on
4	Optional	Get	Status	BOOL	Output point status	0=OK; 1=failure or alarm
5	Optional	Set	Fault Action	BOOL	Action taken on output’s value in Recoverable Fault state	0=Fault Value attribute; 1=hold last state
6	Optional	Set	Fault Value	BOOL	User-defined value for use with Fault State attribute	0=off; 1=on
7	Optional	Set	Idle Action	BOOL	Action taken on output’s value in Recoverable Fault state	0=Idle Value attribute; 1=hold last state
8	Optional	Set	Idle Value	BOOL	User-defined value for use with Idle State attribute	0=off; 1=on

Discrete Output Point Object, Class Code: 09<sub>Hex</sub>

Attribute ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
9	Optional	Set	Run_Idle_Command	BOOL	Generates the Receive_Idle or Receive_Ready_to_Run event	0=Receive_Idle 1=Receive_Ready_to_Run
10	Optional	Set	Flash	BOOL	Flash output at periodic rate if point is ON	0=no flash; 1=flash
11	Optional	Set	Flash Rate	USINT	Flash Rate for Flash attribute	unsigned positive integer indicating frequency in Hz, e.g. 1= 1Hz
12	Optional	Get	Object State	USINT	State of the object	1 = Non-Existent 2 = Available 3 = Idle 4 = Ready 5 = Run 6 = Recoverable Fault 7 = Unrecoverable Fault 255 = Reserved
13	Conditional <sup>1</sup>		Test Output Mode for Safety	See CIP Safety Specification (Volume 5, Chapter 5)		

1 – This attribute is not allowed if the device is not a safety device. If the device is a safety device, see Volume 5.

**Important:** Optional attributes either provide more information about the discrete output point or alter the behavior of the output point. If the following optional instance attributes are not supported the default attribute values below indicate the required behavior for instances of this object class:

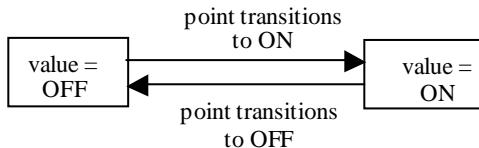
Table 5-10.3 Optional Attribute Default Values

Attribute ID	Name	Default Value
5	Fault Action	0
6	Fault Value	0
7	Idle Action	0
8	Idle Value	0
10	Flash	0

### 5-10.2.1 Value

The only required Instance attribute is *Value*. The required behavior of a Discrete Output Point's value is that it outputs a boolean value of OFF or ON.

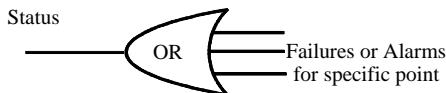
**Figure 5-10.4 Discrete Output Point Object Value Attribute Illustration**



### 5-10.2.2 Status

The *Status* attribute is simply a logical OR of all possible failure or alarm conditions for the point.

**Figure 5-10.5 Discrete Output Point Status Attribute**



### 5-10.2.3 Fault and Idle Attributes

These optional attributes define a safe state for the DOP when in the Idle or Recoverable Fault states:

- Fault Action
- Fault Value
- Idle Action
- Idle Value

**Table 5-10.6 Fault/Idle Action Pair Definition**

This attribute pair	Defines
Fault Action and Fault Value	the value of the DOP in the Fault state.
Idle Action and Idle Value	the value of the DOP in the Idle state.

The following dependencies exist among these attributes:

**Table 5-10.7 Fault/Idle Action Pair Dependencies**

If this attribute is supported	Then this attribute must also be supported by definition
Fault Action	Fault Value (although this attribute could be defined to always be OFF).
Idle Action	Idle Value (although this attribute could be defined to always be OFF).

The “Action” attributes dictate what the DOP will do upon entering that state. The DOP will either hold its value in the last state or update it to the value stored in the corresponding Fault or Idle Value attribute.

**Discrete Output Point Object, Class Code: 09<sub>Hex</sub>**

Upon entering the Recoverable\_Fault state the DOP will behave according to the following table.

**Table 5-10.8 Recoverable Fault State Behavior**

	<b>Fault_State = 0</b>	<b>Fault_State = 1</b>
Fault_Value = 0	DOP uses the value in the Fault_Value attribute (0) to update its value.	DOP leaves Value in last state. Fault_Value attribute has no affect.
Fault_Value = 1	DOP uses the value in the Fault_Value attribute (1) to update its value.	DOP leaves Value in last state. Fault_Value attribute has no affect.

The Idle attributes follow similar behavior.

**Important:** There is one deviation from the behavior specified in the table above. If the DOP enters the Recoverable\_Fault state from the Idle state in response to the I/O connection transitioning to Timed Out, the DOP's value should go unchanged. This is shown in the DOP's State Transition Diagram.

**5-10.2.4 Run\_Idle Command**

The *Run\_Idle Command* attribute causes the Receive\_Ready\_to\_Run or Receive\_Idle event to be sent to the DOP. Refer to the DOP's State Transition Diagram to see the resulting transitions. This attribute only has effect when the Discrete Output object is in the Idle, Ready, or Run states. While in the Available or Recoverable Fault state, an attempt to set this attribute will result in an Object\_State\_Conflict error. A read (Get\_Attribute\_Single) of this attribute will result in a zero being returned always.

**5-10.2.5 Flash**

The *Flash* attribute modifies the behavior of the DOP such that when the DOP is in the ON state, the output flashes OFF and ON at a periodic rate.

**5-10.2.6 Flash Rate**

The *Flash Rate* attribute modifies the behavior of the DOP in that when the DOP is in the ON state and the Flash attribute is ON, it sets the periodic flash rate by an unsigned positive integer.

**5-10.3 Common Services**

The Discrete Output Object provides the following Common Services:

**Table 5-10.9 Discrete Output Point Object Common Services**

<b>Service</b>	<b>Need in Implementation</b>		<b>Service Name</b>	<b>Description of Service</b>
	<b>Code</b>	<b>Class</b>		
0Ehex	Conditional*	Required	Get_Attribute_Single	Returns the contents of the specified attribute.
10hex	n/a	Required	Set_Attribute_Single	Modifies an attribute value.
01hex	Optional	Optional	Get_Attributes_All	Returns a predefined listing of this objects attributes (See the Get_Attributes_All Response definition below)

Discrete Output Point Object, Class Code: 09<sub>Hex</sub>

Service Code	Need in Implementation		Service Name	Description of Service
	Class	Instance		
02hex	n/a	Optional	Set_Attributes_All	Modifies the contents of the attributes of the class or object.

\*The Get\_Attribute\_Single service is REQUIRED if any attributes are implemented.

See the Appendix A for definitions of these common services.

## 5-10.3.1

**Get\_Attributes\_All Response**

At the Class level, the order of the attributes returned in the “Object/service specific reply data” portion (see Chapter 4 of Volume I for a description of the Service Data field) of the Get\_Attributes\_All response is as follows:

**Table 5-10.10 Get\_Attributes\_All Response Data – Class Level**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0								Revision (low byte) Default = 1
1								Revision (high byte) Default = 0
2								Max Instance (low byte) Default = 0
3								Max Instance (high byte) Default = 0
4								Max ID Number of Class Attributes (low byte) Default = 0
5								Max ID Number of Class Attributes (high byte) Default = 0
6								Max ID Number of Instance Attributes (low byte) Default = 3
7								Max ID Number of Instance Attributes (high byte) Default = 0

**Important:** Insert default values for all unsupported attributes.

At the Instance level, the order of the attributes returned in the “Object/service specific reply data” portion of the Get\_Attributes\_All response is as follows:

**Table 5-10.11 Get\_Attributes\_All Response Data – Instance Level**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0								Number of Attributes Default = 0
1								Attribute List (attribute #1)
n								Attribute List (attribute #m)
n+1	0	0	0	0	0	0		Value
n+2	0	0	0	0	0	0		Status Default = 0
.	0	0	0	0	0	0		Fault State Default = 0
.	0	0	0	0	0	0		Fault Value Default = 0
.	0	0	0	0	0	0		Idle State Default = 0
.	0	0	0	0	0	0		Idle Value Default = 0
.	0	0	0	0	0	0		Flash Default = 0
.								Flash Rate Default = 0
.								Object State Default = 255

**Important:** If the instance attribute ”Number of Attributes” is not supported, the default value of zero is to be inserted in its place and **no** members of the ”Attribute List” attribute will follow.

**Discrete Output Point Object, Class Code: 09<sub>Hex</sub>**

**Important:** Insert default values for all unsupported attributes.

**5-10.3.2 Set\_Attributes\_All Request**

No settable attributes currently exist at the Class level for the Discrete Output Point object.

At the Instance level, the order of attributes passed in the Set\_Attributes\_All request is as follows:

**Table 5-10.12 Set\_Attributes\_All Request Data – Instance Level**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	0	0	0	Value
1	0	0	0	0	0	0	0	Fault Action
2	0	0	0	0	0	0	0	Fault Value
3	0	0	0	0	0	0	0	Idle State
4	0	0	0	0	0	0	0	Idle Value
5	0	0	0	0	0	0	0	Flash
6	Flash Rate							

**Important:** The Set\_Attributes\_All service is to be supported only if **all** settable attributes shown above are implemented as settable.

#### **5-10.4 Object-specific Services**

The Discrete Output Point Object provides no Object-specific services.

#### **5-10.5 Behavior**

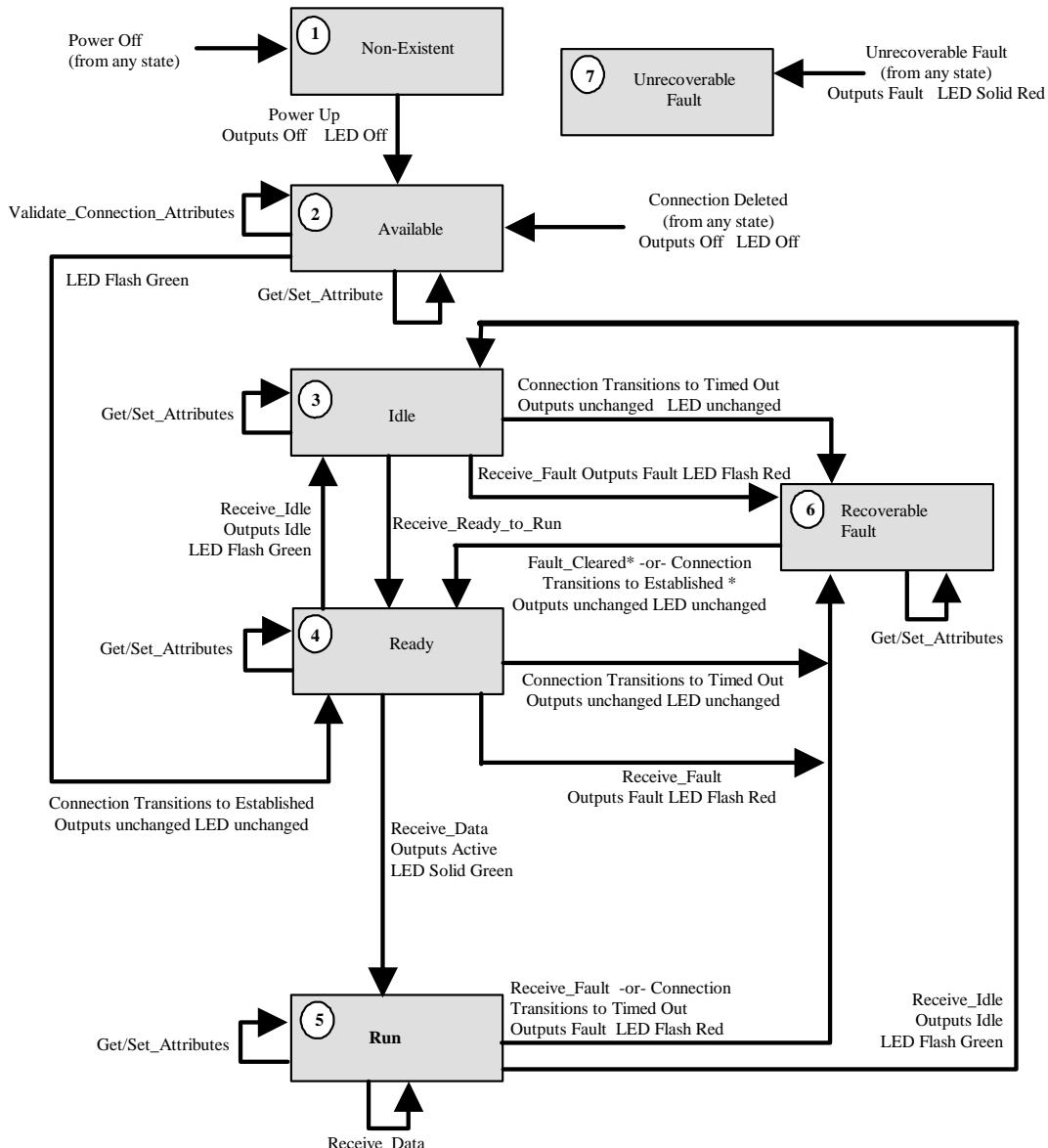
The State Transition Diagram in Figure 5-10.5.1 provides a graphical description of the events and corresponding state transitions. A subset of the states and events may be supported in an application, but the behavior must be consistent.

The State Event Matrix in this section lists all pertinent events and the corresponding action to be taken while in each state.

**Important:** Events can occur simultaneously, but the *Fault* events have priority if they occur simultaneously with other events. In addition, if the Receive\_Data event occurs simultaneously with any other event, the other event takes precedence.

Discrete Output Point Object, Class Code: 09<sub>Hex</sub>

Figure 5-10.13 State Transition Diagram for Discrete Output Point Object



\*And no other faults exist

Note: LED = I/O Object Status LED

The SEM contains the following states:

- **Non-Existent:** module without power.
- **Available:** DOP defaults configured, waiting for connection.
- **Idle:** DOP in Idle mode and does not apply received data.
- **Ready:** waiting for valid data to apply.
- **Run:** DOP applying received data to its output.
- **Recoverable Fault:** a recoverable fault has occurred.
- **Unrecoverable Fault:** an unrecoverable fault has occurred.

**Discrete Output Point Object, Class Code: 09<sub>Hex</sub>**

The SEM also contains these events:

**Table 5-10.14 Discrete Output Point Event Definitions**

<b>This event</b>	<b>Is</b>
Receive_Data	an event that signals the reception of I/O data and causes the object to transition to the Run state.
Receive_Fault	an event that is internally generated and product-specific. The network does not know if a Fault has occurred.
Receive_Idle	the setting of the Run_Idle Command attribute to the value 0 -or- the IO connection object receives an I/O message <b>containing no application data</b>
Receive_Run	the setting of the Run_Idle Command attribute to the value 1.
Apply_Attributes	the Apply service of the I/O connection object the Discrete Output Point object is connected to. Note: the application is responsible for validating the connection object's attributes.
Connection Deleted	I/O connection deleted.
Connection Transitions to Established	I/O connection transitions to Established.
Connection transitions to the Timed Out state	the expiration of the connection timer.

Discrete Output Point Object, Class Code: 09<sub>Hex</sub>

Table 5-10.15 State Event Matrix for the Discrete Output Point Object

Event	State						
	Non - Existent	Available	Idle	Ready	Run	Recoverable Fault	Unrecoverable Fault
Receive_Data	Not applicable	Not applicable	If data length is non-zero transition to Run Verify and Accept Data Transition to Run Ignore event	LED Solid Green, Accept Data, Transition to Run	If data length is zero <sup>2</sup> Transition to Idle Otherwise Verify and Accept Data	Ignore event	Ignore event
Receive_Fault	Not applicable	Not applicable	Transition to Recoverable Fault <sup>1</sup>	Transition to Recoverable Fault <sup>1</sup>	Transition to Recoverable Fault <sup>1</sup>	Ignore event	Ignore event
Receive_Idle	Not applicable	Return Error (object state conflict)	Ignore event	Transition to Idle <sup>2</sup>	Transition to Idle <sup>2</sup>	Return Error (object state conflict)	Ignore event
Receive_Ready_to_Run	Not applicable	Return Error (object state conflict)	Transition to Ready	Ignore event	Ignore event	Return Error (object state conflict)	Ignore event
Validate_Connection_Attributes	Not applicable	Verify attributes, return results	Return Error (object state conflict)	Return Error (object state conflict)	Return Error (object state conflict)	Return Error (object state conflict)	Ignore event
Connection Deleted	Not applicable	Ignore event	Transition to Available	Transition to Available	Transition to Available	Transition to Available	Ignore event
Connection Transition to Established	Not applicable	Transition to Ready	Return Error (object state conflict)	Return Error (object state conflict)	Return Error (object state conflict)	Transition to Ready <sup>3</sup>	Ignore event
Connection transitions to Timed Out state	Not applicable	Not applicable	Transition to Recoverable Fault	Transition to Recoverable Fault	Transition to Recoverable Fault <sup>1</sup>	Ignore event	Ignore event
Get_Attribute	Return Error (Object Does Not Exist)	Return value	Return value	Return value	Return value	Return value	Ignore event
Set_Attribute	Return Error (Object Does Not Exist)	Verify, and Accept, and Apply value	Verify, and Accept, and Apply value	Verify, and Accept, and Apply value	Verify, and Accept, and Apply value	Verify, and Accept, and Apply value	Ignore event
Fault_Cleared	Ignore event	Ignore event	Ignore event	Ignore event	Ignore event	Transition to Ready <sup>3</sup>	Ignore event

<sup>1</sup> Fault: Hold Last State OR use Fault Value.<sup>2</sup> Idle: Hold Last State OR use Idle Value.<sup>3</sup> If no other faults exist (Note, a Connection time out is considered a fault)

## 5-10.5.1 Attribute Access

Except in the Non-Existent and Unrecoverable Fault states, all attributes are gettable or settable according to their access rules.

## 5-11 Analog Input Point Object

**Class Code: 0A<sub>hex</sub>**

The Analog Input Point (AIP) Object models analog inputs in a product. It can be used in applications as simple as a single analog point and as complex as an analog I/O control module. Note that the term "input" is defined from the network's point of view. An input will produce data on the network.

The Analog Input Point interface is to real input points such as a thermocouple or pressure transducer. The input is sampled and the data is stored in this object's VALUE attribute. A sample of the analog input value is triggered via an external command (input change-of-state, cyclic data trigger, etc.)

### 5-11.1 Revision History

Since the initial release of this object class definition changes have been made that require a revision update of this object class. The table below represents the revision history:

**Table 5-11.1 Analog Input Point Object Revision History**

Revision	Reason for object definition update
01	Initial Definition at First Release of Specification
02	IDLE state removed from this object's behavior

### 5-11.2 Class Attributes

**Table 5-11.2 Analog Input Point Object Class Attributes**

Attr ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1	Required	Get	Revision	UINT	Revision of this object	The current value assigned to this attribute is two (02).
2 thru 7	These class attributes are optional and are described in Chapter 4 of this specification.					

### 5-11.3 Instance Attributes

Table 5-11.3 Analog Input Point Object Instance Attributes

Attr ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1	Optional	Get	Number of Attributes	USINT	Number of attributes supported	
2	Optional	Get	Attributes List	Array of USINT	List of attributes supported by the point	
3	Required	Get	Value	INT or based on attribute 8	Analog input value. The data type defaults to INT but may be changed based on attribute 8.	
4	Optional	Get	Status	BOOL	Indicates if a fault or alarm has occurred.	0= operating without alarms or faults. 1=alarm or fault condition exists, the Value attribute may not represent the actual field value.
5	Optional	Get	Owner Vendor ID	UINT	Vendor ID of channel's owner	
6	Optional	Get	Owner Serial Number	UDINT	32-bit serial number of channel's owner	
7	Optional	Set	Input Range	USINT	Input range the point is operating in	0 = -10V to 10V 1 = 0V to 5V 2 = 0V to 10V 3 = 4mA to 20mA 4 = -15mV to 75mV 5 = -15mV to 30mV 6 = -5V to 5V 7 = 1V to 5V 8 = 0mA to 20mA 9 = 0mA to 50mA 10 – 99 = Reserved 100-131=Vendor Specific 132-154 =Reserved 255 = Only return with Get_Attributes_All if attribute not supported
8	Optional	Set	Value Data Type	USINT	Determines the data type of Value	0 = INT 1 = REAL 2 = USINT 3 = SINT 4 = DINT 5 = LINT 6 = UINT 7 = UDINT 8 = ULINT 9 = LREAL 100 = vendor specific

## 5-11.4 Common Services

The Analog Input Object provides the following Common Services:

**Table 5-11.4 Analog Input Point Object Common Services**

Service Code	Need in Implementation		Service Name	Description of Service
	Class	Instance		
0Ehex	Required	Required	Get_Attribute_Single	Returns the contents of the specified attribute.
10hex	n/a	Conditional <sup>1</sup>	Set_Attribute_Single	Modifies an attribute value.
01hex	Optional	Optional	Get_Attributes_All	Returns a predefined listing of this objects attributes (See the Get_Attributes_All Response definition below)
02hex	n/a	Optional	Set_Attributes_All	Modifies the contents of the attributes of the class or object.

1 The Set\_Attribute\_Single service is required only if Instance Attributes 7 and/or 8 are implemented.

See Appendix A for definition of these services

### 5-11.4.1 Get\_Attributes\_All Response

At the Class level, the order of the attributes returned in the “Object/service specific reply data” portion of the Get\_Attributes\_All response is as follows:

**Table 5-11.5 Get\_Attributes\_All Response Data – Class Level**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0					Revision (low byte) Default = 2			
1					Revision (high byte) Default = 0			
2					Max Instance (low byte) Default = 0			
3					Max Instance (high byte) Default = 0			
4					Max ID Number of Class Attributes (low byte) Default = 0			
5					Max ID Number of Class Attributes (high byte) Default = 0			
6					Max ID Number of Instance Attributes (low byte) Default = 3			
7					Max ID Number of Instance Attributes (high byte) Default = 0			

**Important:** Insert default values for all unsupported attributes.

**Analog Input Point Object, Class Code: 0A<sub>Hex</sub>**

At the Instance level, the order of the attributes returned in the “Object/service specific reply data” portion (see Chapter 4 for a description of the Service Data field) of the Get\_Attributes\_All response is as follows:

**Table 5-11.6 Get\_Attributes\_All Response Data – Instance Level**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Number of Attributes Default = 0							
1	Attribute List (attribute #1)							
n	Attribute List (attribute #m)							
n+1	Value Data Type Default = 0							
.	Value (low byte)							
.	Value (high byte)							
.	0	0	0	0	0	0	0	Status Default = 0
.	Owner Vendor ID (low byte) Default = 0							
.	Owner Vendor ID (high byte) Default = 0							
.	Owner Serial Number (low byte) Default = 0							
.	Owner Serial Number Default = 0							
.	Owner Serial Number Default = 0							
.	Owner Serial Number (high byte) Default = 0							
.	Input Range Default = 255 *							

The default value of 255 must be returned for the Input Range attribute ONLY if it is not supported by the device.

**Important:** Insert default values for all unsupported attributes.

**Important:** If the instance attribute ”Number of Attributes” is not supported, the default value of zero is to be inserted in its place and **no** members of the ”Attribute List” attribute will follow.

**5-11.4.2 Set\_Attributes\_All Request**

No settable attributes currently exist at the Class level for the Analog Input Point object.

At the Instance level, the order of attributes passed in the Set\_Attributes\_All request is as follows:

**Table 5-11.7 Set\_Attributes\_All Request Data – Instance Level**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Input Range							
1	Value Data Type							

**Important:** The Set\_Attributes\_All service is to be supported only if **all** settable attributes shown above are implemented as settable.

## 5-11.5 Object-specific Services

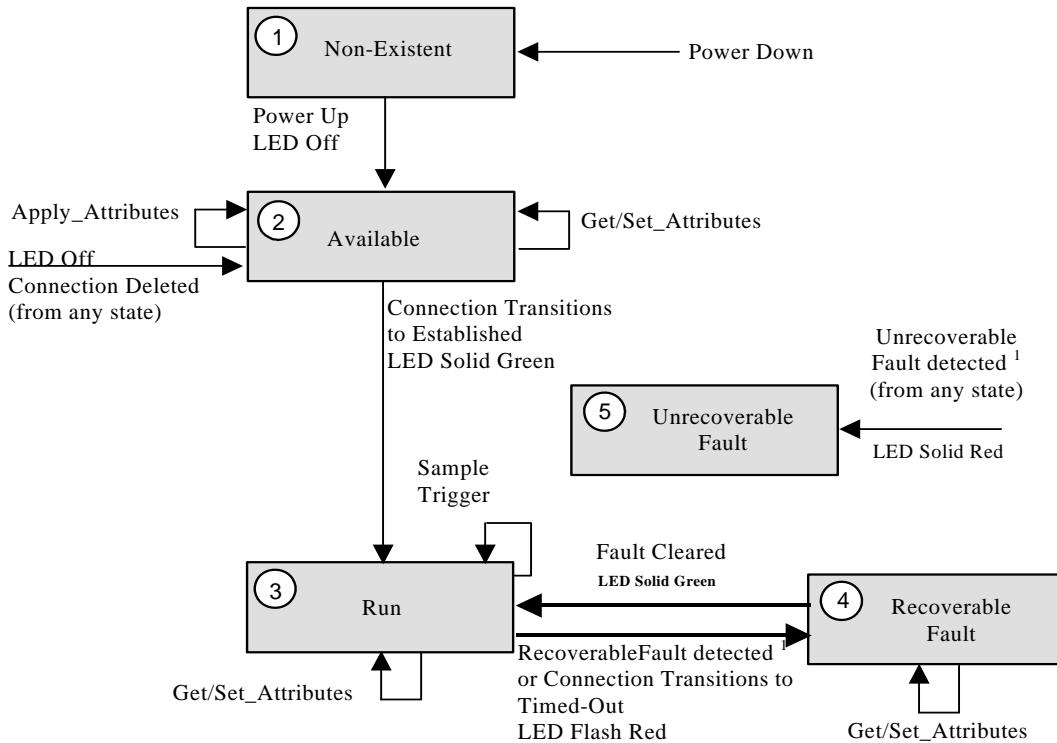
The Analog Input Object provides no Object-specific services:

## 5-11.6 Behavior

The State Transition Diagram in Figure 5-11.8 provides a graphical description of the events and corresponding state transitions. A subset of the states and events may be supported in an application, but the behavior must be consistent.

The State Event Matrix in this section lists all pertinent events and the corresponding action to be taken while in each state. A subset of the states and events may be supported in an application, but the behavior must still be consistent.

**Figure 5-11.8 State Transition Diagram for Analog Input Point Object**



1. Indicated by the application      LED = I/O Status LED

**Important:** Events can occur simultaneously, but the *Fault* events have priority if they occur simultaneously with other events.

The SEM contains the following states:

- **Non-Existent:** module with no power.
- **Available:** waiting for a connection, power-up analog input point defaults are set.
- **Run:** AIP sensing data from its input and transmitting the data.
- **Recoverable Fault:** a recoverable fault has occurred.
- **Unrecoverable Fault:** an unrecoverable fault has occurred.

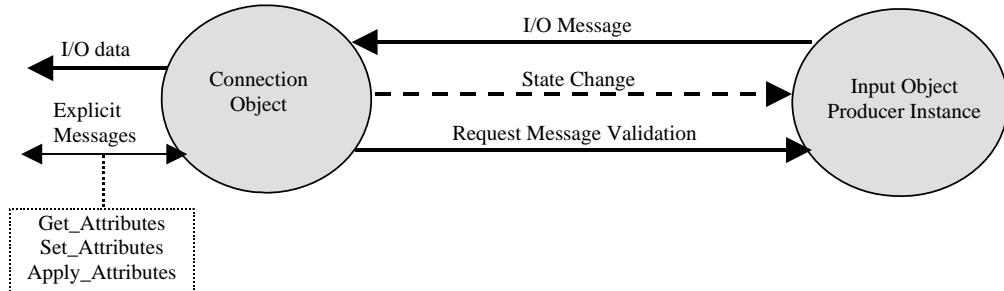
**Analog Input Point Object, Class Code: 0A<sub>Hex</sub>**

The SEM also contains these events:

**Table 5-11.9 Analog Input Point Object Event Definitions**

This event	Is
Sample Trigger	a change of state; cyclic timer trigger; non-zero length Bit Strobe or Poll Command.
Connection Deleted	I/O connection deleted.
Apply_Attributes	the Apply service of the I/O connection object the Analog Input Point object is connected to. Note: the application is responsible for validating the connection object's attributes.
Fault Cleared	the application clearing a detected fault
Connection Transitions to Established	I/O connection transitions to Established.
Connection Transitions to Timed Out state	I/O connection transitions to Timed-Out

Figure 5-11.10 shows a conceptual illustration of the state machine for a typical input object (producing application). The events listed above are represented by the dotted line labeled “state change.”

**Figure 5-11.10 State Machine Illustration for Typical Analog Input Point Object**

**Table 5-11.11 State Event Matrix for the Analog Input Point Object**

Event	State				
	Non-Existent	Available	Run	Recoverable Fault	Unrecoverable Fault
Sample Trigger	Not Applicable	Ignore event	Sample data, Send data	Ignore event	Ignore event
Apply Attributes	Not Applicable	Verify attributes, return result	Return error (Object State Conflict)	Return error (Object State Conflict)	Ignore event
Connection Deleted	Not Applicable	Ignore Event	Transition to Available	Transition to Available	Ignore event
Connection Transitions to Established	Not Applicable	Transition to Run	Ignore event	Ignore event	Ignore event
Connection Transitions to Timed Out state	Not Applicable	Ignore event	Transition to Recoverable Fault	Ignore event	Ignore event
Fault Cleared	Not Applicable	Not Applicable	Not Applicable	Transition to Run	Ignore event
Get_Attribute	Return Error (Object Does Not Exist)	Return value	Return value	Return value	Ignore event
Set_Attribute	Return Error (Object Does Not Exist)	Accept value	Accept value	Accept value	Ignore event
I/O Status LED	Off	Flash Green	Solid Green	Flash Red	Solid Red

### 5-11.6.1 Attribute Access Rules

Except in the Non-Existent and Unrecoverable Fault states, all attributes are gettable or settable according to their access rules.

Because the only required Instance attribute is *Value*, the only required behavior of an AIP is that it indicates an analog value.

Optional attributes either provide more information about the AIP or alter the behavior of the input point.

#### 5-11.6.1.1 Status Attribute

The optional *Status* attribute is simply a logical OR of all possible failure or alarm conditions for the point.

**Figure 5-11.12 Analog Input Point Object Status Attribute Operation**

#### **5-11.6.1.2 Owner Vendor ID, Owner Serial Number**

These attributes may be implemented in both the point and group objects. If implemented in both, the values shall be the same for the point objects and the group object that binds the point object. If a connection to this object is made using the Connection Manager, then the value for the Owner Vendor Id and Owner Serial Number shall be the Originator Vendor Id and Originator Serial Number contained in the Forward\_Open request. These values shall be cleared when the connection is closed or deleted.

#### **5-11.6.1.3 Input Range**

The *Input Range* attribute determines the set of analog values within which the inputs may operate. The value of the Input Range affects the default and legal values that other attributes may have.

Input Range is necessary only if the point may be switched to operate within different ranges, which changes the behavior of the point. For example, if the point is configured to operate from 0V to 5V or from 4mA to 20mA, the variability of ranges will likely change the expected behavior of the Value attribute and perhaps change vendor specific attributes.

#### **5-11.6.1.4 Value Data Type**

The *Value Data Type* attribute determines the data type to be used by the attribute Value (and perhaps other attributes as well). If Value Data Type is not used, then Value defaults to INT. Value (and other attributes) may behave as REAL, USINT, or any other data type based upon the definition of Value Data Type, which ultimately provides the ability for the analog point to be defined in any length necessary.

## 5-12 Analog Output Point Object

**Class Code: 0B<sub>hex</sub>**

The Analog Output Point (AOP) models the point level attributes and services of the analog outputs in a product. It can be used to model voltage output or parts of an analog I/O control module. Note that the term "output" is defined from the network's point of view. An output will consume data from the network. The Analog Output Point interface is to real output points such as a Motor Operated Valve (MOV) or Linear Positioner. The output is read from this object's VALUE attribute and applied to the output terminal (e.g. screw terminal).

### 5-12.1 Class Attributes

**Table 5-12.1 Analog Output Point Object Class Attributes**

Attr ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1	Required	Get	Revision	UINT	Revision of this object	The current value assigned to this attribute is two (02).
2 thru 7	These class attributes are optional and are described in Chapter 4 of this specification.					

## 5-12.2 Instance Attributes

Table 5-12.2 Analog Output Point Object Instance Attributes

Attr ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1	Optional	Get	Number of Attributes	USINT	Number of attributes supported	0–255
2	Optional	Get	Attribute List	Array of USINT	List of attributes supported by the point	
3	Required	Set	Value	INT or based on attribute 8	Analog output value. The data type defaults to INT but may be changed based on attribute 8.	
4	Optional	Get	Status	BOOL	Indicates if a fault or alarm has occurred.	0= operating without alarms or faults. 1=alarm or fault condition exists, the Value attribute may not represent the actual field value.
5	Optional	Get	Owner - Vendor ID	UINT	Vendor ID of channel's owner	
6	Optional	Get	Owner - Serial Number	UDINT	32-bit serial number of channel's owner	
7	Optional	Set	Output Range	USINT	Specifies the output range the output channel is to use	0 = 4mA to 20mA 1 = 0V to 10V 2 = 0mA to 20mA 3 = -10V to 10V 4 = 0V to 5V 5 = -5V to 5V 6 = 1V to 5V 7 - 8 = Reserved 9 = 0mA to 50mA 10 – 99 = Reserved 100-131=Vendor Specific 132 – 254 = Reserved 255 = Only return with Get_Attributes_All if attribute not supported
8	Optional	Set	Value Data Type	USINT	Determines the data type of Value	0 = INT (default) 1 = REAL 2 = USINT 3 = SINT 4 = DINT 5 = LINT 6 = UINT 7 = UDINT 8 = ULINT 9 = LREAL 100 = vendor specific

Analog Output Point Object, Class Code: 0B<sub>Hex</sub>

Attr ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
9	Optional	Set	Fault Action	USINT	Output value to go to on failure or fault	0 = hold last state (default) 1 = low limit <sup>1</sup> 2 = high limit 3 = use Fault Value 4 - 99 = reserved 100 - 199 = vendor specific 200 - 299 = reserved
10	Optional	Set	Idle Action	USINT	Output value to go to on idle mode	0 = hold last state (default) 1 = low limit <sup>1</sup> 2 = high limit 3 = use Idle Value 4 - 99 = reserved 100 - 199 = vendor specific 200 - 299 = reserved
11	Required if Fault Action is implemented	Set	Fault Value	INT or based on attribute 8	User defined value outputs go to in fault mode if Fault State = 3, user specified value	
12	Optional	Set	Idle Value	INT or based on attribute 8	User defined value outputs go to in idle mode if Idle State = 3, user specified value	
13	Required if Idle Action is implemented	Set	Command	BOOL	Changes state of AOP to Idle Mode or Run Mode	0 = idle 1 = run
14	Optional	Get	Object State	USINT	State of the object	1 = Non-Existent 2 = Available 3 = Idle 4 = Ready 5 = Run 6 = Recoverable Fault 7 = Unrecoverable Fault 255 = Reserved

<sup>1</sup> All values 0, 1, 2, and 3 shall be supported when the Fault Action or Idle Action attributes are implemented

**Important:** If the following optional instance attributes are not supported, the default attribute values below indicate the required behavior for instances of this object class:

Table 5-12.3 Analog Output Point Default Values for Unsupported Attributes

Attribute ID	Name	Default Value
7	Output Range	0
8	Value Data Type	0
9	Fault State	0
10	Idle State	0
11	Fault Value	minimum value of range (attribute 7)
12	Idle Value	0

### 5-12.3 Common Services

The Analog Output Object provides the following Common Services:

**Table 5-12.4 Analog Output Point Object Common Services**

Service Code	Need in Implementation		Service Name	Description of Service
	Class	Instance		
0Ehex	Conditional*	Required	Get_Attribute_Single	Returns the contents of the specified attribute.
10hex	n/a	Required	Set_Attribute_Single	Modifies an attribute value.
01hex	Optional	Optional	Get_Attributes_All	Returns a predefined listing of this objects attributes (See the Get_Attributes_All Response definition below)
02hex	n/a	Optional	Set_Attributes_All	Modifies the contents of the attributes of the class or object.

\*The Get\_Attribute\_Single service is REQUIRED if any class attributes are implemented.

See the Appendix A for definitions of these common services.

#### 5-12.3.1 Get\_Attributes\_All Response

At the Class level, the order of the attributes returned in the “Object/service specific reply data” portion of the Get\_Attributes\_All response is as follows:

**Table 5-12.5 Get\_Attributes\_All Response Data – Class Level**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0								Revision (low byte) Default = 1
1								Revision (high byte) Default = 0
2								Max Instance (low byte) Default = 0
3								Max Instance (high byte) Default = 0
4								Max ID Number of Class Attributes (low byte) Default = 0
5								Max ID Number of Class Attributes (high byte) Default = 0
6								Max ID Number of Instance Attributes (low byte) Default = 3
7								Max ID Number of Instance Attributes (high byte) Default = 0

**Important:** Insert default values for all unsupported attributes.

At the Instance level, the order of the attributes returned in the “Object/service specific reply data” portion of the Get\_Attributes\_All response is as follows:

Analog Output Point Object, Class Code: 0B<sub>Hex</sub>

Table 5-12.6 Get\_Attributes\_All Response Data – Instance Level

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Value Data Type, default = 0 = INT							
1	Value (low byte)							
n	Value (high byte)							
n+1	Number of Attributes Default = 0							
.	Attribute List (attribute #1)							
.	Attribute List (attribute #m)							
.	0	0	0	0	0	0	0	Status Default = 0
.	Owner Vendor ID (low byte) Default = 0							
.	Owner Vendor ID (high byte) Default = 0							
.	Owner Serial Number (low byte) Default = 0							
.	Owner Serial Number Default = 0							
.	Owner Serial Number (high byte) Default = 0							
.	Output Range Default = 255							
.	Fault Action Default = 0							
.	Idle Action Default = 0							
.	Fault Value (low byte)							
.	Fault Value (high byte)							
.	Idle Value (low byte)							
.	Idle Value (high byte)							
.	0	0	0	0	0	0	0	Command Default = 0
.	Object State Default = 255							

**Important:** Insert default values for all unsupported attributes.

The default value of 255 must be returned for the Output Range attribute ONLY if it is not supported by the device.

**Important:** If the instance attribute "Number of Attributes" is not supported, the default value of zero is to be inserted in its place and no members of the "Attribute List" attribute will follow.

### 5-12.3.2 Set\_Attributes\_All Request

No settable attributes currently exist at the Class level for the Analog Output Point object.

At the Instance level, the order of attributes passed in the Set\_Attributes\_All request is as follows:

Analog Output Point Object, Class Code: 0B<sub>Hex</sub>

Table 5-12.7 Set\_Attributes\_All Request Data – Instance Level

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0								Value Data Type
1								Value (low byte)
n								Value (high byte)
N+1								Output Range
.								Fault Action
.								Idle Action
.								Fault Value (low byte)
.								Fault Value (high byte)
.								Idle Value (low byte)
.								Idle Value (high byte)
.	0	0	0	0	0	0	0	Command

**Important:** The Set\_Attributes\_All service is to be supported only if **all** settable attributes shown above are implemented as settable.

#### 5-12.4 Object-specific Services

The Analog Output Point Object provides no Object-specific services.

#### 5-12.5 Behavior

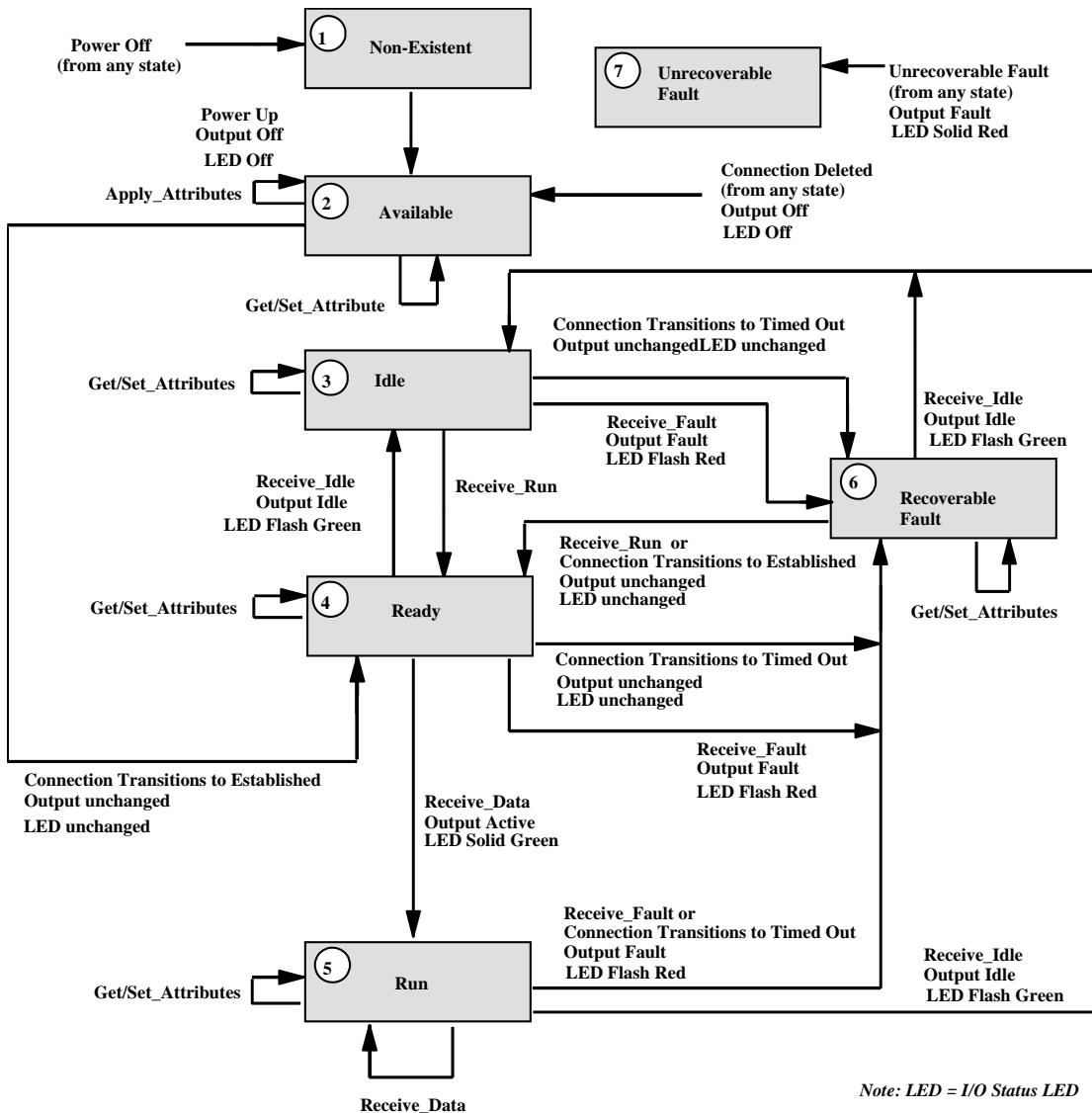
The State Transition Diagram in Figure 5-12.8 provides a graphical description of the events and corresponding state transitions. A subset of the states and events may be supported in an application, but the behavior must be consistent.

The State Event Matrix in Table 5-12.9 lists all pertinent events and the corresponding action to be taken while in each state. A subset of the states and events may be supported in an application, but the behavior must still be consistent.

**Important:** Events can occur simultaneously, but the *Fault* events have priority if they occur simultaneously with other events. In addition, if the Receive\_Data event occurs simultaneously with any other event, the other event takes precedence.

Analog Output Point Object, Class Code: 0B<sub>Hex</sub>

Figure 5-12.8 State Transition Diagram for Analog Output Point Object



The following SEM contains these states:

- **Non-Existent:** module without power.
- **Available:** AOP defaults configured, waiting for connection.
- **Idle:** AOP in standby mode and does not apply received data.
- **Ready (to run):** waiting for valid data to apply.
- **Run:** AOP applying received data to its output.
- **Recoverable Fault:** a recoverable fault has occurred.
- **Unrecoverable Fault:** an unrecoverable fault has occurred.

Analog Output Point Object, Class Code: 0B<sub>Hex</sub>

The SEM also contains these events:

**Table 5-12.9 Analog Output Point Object Event Definitions**

This event	Is
Receive_Data	an event that signals the reception of I/O data and causes the object to transition to the Run state.
Receive_Fault	an event that is internally generated and product-specific. The network does not know if a Fault has occurred.
Receive_Idle	the setting of the COMMAND attribute to the value 0 -or- within the Master/Slave paradigm the event signaled when the I/O connection object receives a Bit-Strobe or Poll Command message <i>containing no application data</i>
Receive_Run	the setting of the COMMAND attribute to the value 1 -or- within the Master/Slave paradigm the event signaled when the I/O connection object receives a Bit-Strobe or Poll Command message <i>containing application data</i>
Apply_Attributes	the Apply service of the I/O connection object the Analog Output Point object is connected to. Note: the application is responsible for validating the connection object's attributes.
Connection Deleted	I/O connection deleted.
Connection Transitions to Established	I/O connection transitions to Established.
Connection transitions to the Timed Out state	the expiration of the connection timer.

The figure below is a conceptual illustration of the state machine for a typical output object (consuming application). The events listed above are represented by the dotted line labeled “state change.”

Analog Output Point Object, Class Code: 0B<sub>Hex</sub>

Figure 5-12.10 State Machine Illustration for Typical Analog Output Point Object

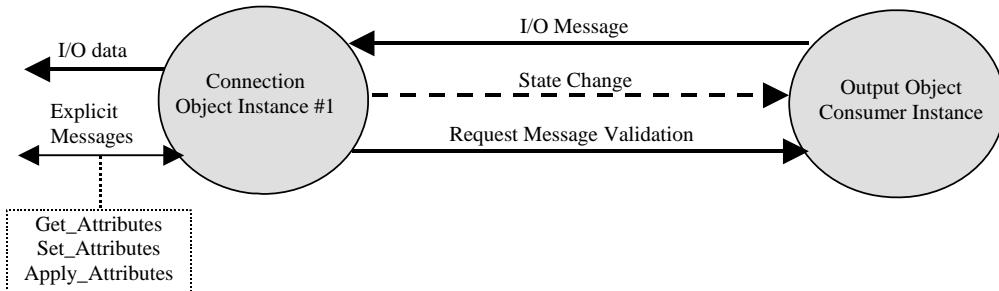


Table 5-12.11 State Event Matrix for the Analog Output Point Object

Event	State							
	Non-Existent	Available	Idle	Ready	Run	Recoverable Fault	Unrecoverable Fault	
Receive_Data	Not applicable	Not applicable	Ignore event	LED Solid Green, Accept Data, Transition to Run	Accept Data	Ignore event	Ignore event	
Receive_Fault	Not applicable	Not applicable	Transition to Recoverable Fault <sup>1</sup>	Transition to Recoverable Fault <sup>1</sup>	Transition to Recoverable Fault <sup>1</sup>	Ignore event	Ignore event	
Receive_Idle	Not applicable	Not applicable	Ignore event	Transition to Idle <sup>2</sup>	Transition to Idle <sup>2</sup>	Transition to Idle <sup>2</sup>	Ignore event	
Receive_Run	Not applicable	Not applicable	Transition to Ready	Ignore event	Ignore event	Transition to Ready	Ignore event	
Apply_Attributes	Not applicable	Verify attributes, return results	Return Error (object state conflict)	Return Error (object state conflict)	Return Error (object state conflict)	Return Error (object state conflict)	Ignore event	
Connection Deleted	Not applicable	Ignore event	Transition to Available	Transition to Available	Transition to Available	Transition to Available	Ignore event	
Connection Transition to Established	Not applicable	Transition to Ready	Return Error (object state conflict)	Return Error (object state conflict)	Return Error (object state conflict)	Transition to Ready	Ignore event	
Connection Transitions to Timed Out state	Not applicable	Not applicable	Transition to Recoverable Fault	Transition to Recoverable Fault	Transition to Recoverable Fault <sup>1</sup>	Ignore event	Ignore event	
Get_Attribute	Return Error (Object Does Not Exist)	Return value	Return value	Return value	Return value	Return value	Ignore event	
Set_Attribute	Return Error (Object Does Not Exist)	Accept value	Accept value	Accept value	Accept value	Accept value	Ignore event	

1 - Fault: Hold Last State OR use Fault Value.

2 - Idle: Hold Last State OR use Idle Value.

### 5-12.5.1 Attribute Access Rules

Except in the Non-Existent and Unrecoverable Fault states, all attributes are gettable or settable according to their access rules.

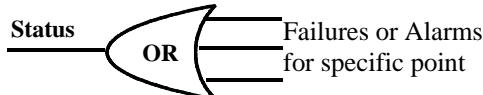
Because the only required Instance attribute is Value, the only required behavior of a Analog Output Point is that it outputs an analog value within the defined range for value (see Instance Attribute 8).

Optional attributes either provide more information about the AOP or alter the behavior of the output point.

#### 5-12.5.1.1 Status Attribute - Optional

The optional Status attribute is simply a logical OR of all possible failure or alarm conditions for the point.

**Figure 5-12.12 Analog Output Point Object Status Attribute Operation**



#### 5-12.5.1.2 Command Attribute – Optional

The Command attribute explicitly controls whether the AOP is in the Run or Idle state. It has effect only when the **Analog** Output object is in the Idle, Ready, Recoverable Fault or Run states. While in the Available state, an attempt to set this attribute will result in an Object\_State\_Conflict error. A read (Get\_Attribute\_Single) of this attribute will result in a zero being returned always.

#### 5-12.5.1.3 Attributes Used to Determine Safe State

These optional attributes define a “safe state” for the analog output point when in the Idle or Fault states:

- Output Fault State
- Output Fault Value
- Output Idle State
- Output Idle Value

**Table 5-12.13 Determining Output Value in Fault and Idle Conditions**

This attribute pair	Defines
Output Fault State and Output Fault Value	the value of the AOP in the Recoverable Fault state
Output Idle State and Output Idle Value	the value of the AOP point when in the Idle state

The following dependencies exist among these attributes:

**Table 5-12.14 Output Fault and Idle State Attribute Dependencies**

If this attribute is supported	Then this attribute must also be supported by definition
Output Fault State	Fault Value (although this attribute could be defined to always be LOW).

Output Idle State

Idle Value

#### **5-12.5.1.4    Owner VendorId, Owner Serial Number**

These attributes may be implemented in both the point and group objects. If implemented in both, the values shall be the same for the point objects and the group object that binds the point object. If a connection to this object is made using the Connection Manager, then the value for the Owner Vendor Id and Owner Serial Number shall be the Originator Vendor Id and Originator Serial Number contained in the Forward\_Open request. These values shall be cleared when the connection is closed or deleted

#### **5-12.5.1.5    Output Range Attribute – Optional**

The Output Range attribute determines the set of analog values within which the outputs may operate. The value of the Output Range affects the default and legal values that other attributes may have.

Output Range is necessary only if the point may be switched to operate within different ranges, which changes the behavior of the point. For example, if the point is configured to operate from -10V to 10V or from 0V to 10V, the variability of ranges will likely affect the low value that outputs go to in safety state (you can use -10 V or 0 V, Output Range).

#### **5-12.5.1.6    Value Data Type Attribute – Optional**

The Value Data Type attribute determines the data type to be used by the attribute Value (and perhaps other attributes as well). If Value Data Type is not used, then Value defaults to INT. Value (and other attributes) may behave as REAL, USINT, or any other data type based upon the definition of Value Data Type, which ultimately provides the ability for the analog point to be defined in any length necessary.

## 5-13 Presence Sensing Object

**Class Code: 0E hex**

This object senses the presence or absence of a real world target.

### 5-13.1 Class Attributes

**Table 5-13.1 Presence Sensing Object Class Attributes**

Attribute ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1 thru 7	These class attributes are optional and are described in Chapter 4 of this specification.					

### 5-13.2 Instance Attributes

**Table 5-13.2 Presence Sensing Object Instance Attributes**

Attr ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1	Required	Get	Output	BOOL	0 = switching element open 1 = switching element closed	See Semantics” section
2	Optional	Get	Number of Attributes	USINT	Number of attributes supported	
3	Optional	Get	Attribute List	Array of USINT	List of attributes supported	
4	Optional	Get	Diagnostic	BOOL	0=good 1=fault	
5	Optional	Set	On Delay	UINT	0 – 65,535 ms	
6	Optional	Set	Off Delay	UINT	0 – 65,535 ms	
7	Optional	Set	One Shot Delay	UINT	0 – 65,535 ms	
8	Optional	Set	Operate Mode	BOOL	0 = output attribute as specified 1 = inversion of output attribute	See Semantics” section
9	Optional	Set	Sensitivity	USINT	0 – 255	See Semantics” section
10	Optional	Get	Target Margin	USINT	1 – 255	See Semantics” section
11	Optional	Get	Background Margin	USINT	0 – 100	See Semantics” section
12	Optional	Set	Min Detect Distance	UINT	0 – 65,535 mm	See Semantics” section
13	Optional	Set	Max Detect Distance	UINT	0 – 65,535 mm *	See Semantics” section
14	Optional	Get	Detect Distance	UINT	0 – 65,535 mm	

\* The Max Detect Distance must be greater than or equal to the Min Detect Distance

### 5-13.2.1 Semantics

#### 5-13.2.1.1 Output

The Output value is determined by the switching element in the device. The state of this value is controlled by the:

- operation mode:      *Light Operate (LO)/Dark Operate (DO)*  
*Normally Closed (NC)/Normally Open (NO)*
- detection of an object (absence or presence)
- timing
- type of device (photoelectric, proximity)

The following table indicates output values for common presence sensing devices.

**Table 5-13.3 Common Output Values for Presence Sensing Devices**

Presence Sensing Device	Output Values	
	Output = On	Output = Off
Diffuse Photoelectric, Retroreflective Photoelectric	light detected (LO) no light detected (DO)	no light detected (LO) light detected (DO)
Through-Beam Photoelectric	light beam detected (LO) light beam interrupted (DO)	light beam interrupted (LO) light beam detected (DO)
Inductive Proximity, Capacitive, Ultrasonic, Limit Switch	object present (NO) object absent (NC)	object absent (NO) object present (NC)

#### 5-13.2.1.2 Operate Mode

Use the Operate (Sensing) Mode to invert the level definition of the Output attribute.

**Table 5-13.4 Common Output Values for Presence Sensing Devices**

For	use the Operate Mode to:
photoelectric sensors,	set Light Operate (LO)/Dark Operate (DO) mode.
proximity sensors,	set Normally Open (NO)/Normally Closed (NC) mode.

#### 5-13.2.1.3 Sensitivity

Sensitivity is the amplification of the signal received by the detection device. Sensitivity is defined with either of two methods (i.e. low, medium, high or a numeric scale). This attribute accommodates both methods by converting the terms into a numeric scale (i.e. 0 = lowest, ... , n = highest).

#### 5-13.2.1.4 Target Margin

Target Margin is the signal strength received by the device when the signal strength is above the device output switching threshold. The margin value is a scale of 1 to 255 with 1 representing the device output switching threshold.

#### 5-13.2.1.5 Background Margin

Background Margin is the signal strength received by the device when the signal strength is below the device output switching threshold. The margin value is a scale of 0 to 100 where the value 100 equals the device switching threshold and the value 0 represents no signal received.

#### **5-13.2.1.6 Minimum and Maximum Detect Distances**

The Minimum and Maximum Detect Distances define the range in which an object can be detected. If an object passes through the sensing device's field of view outside this range, the sensing device does not change the output (attribute number 1) of the device to indicate detection of that object.

### **5-13.3 Common Services**

The Presence Sensing Object provides the following Common Services:

**Table 5-13.5 Presence Sensing Object Common Services**

<b>Service Code</b>	<b>Need in Implementation</b>		<b>Service Name</b>	<b>Description of Service</b>
	<b>Class</b>	<b>Instance</b>		
0Ehex	Conditional*	Required	Get_Attribute_Single	Returns the contents of the specified attribute.
10hex	n/a	Optional	Set_Attribute_Single	Modifies an attribute value.

\*The Get\_Attribute\_Single service is REQUIRED if any attributes are implemented.

See the Appendix A for definitions of these common services.

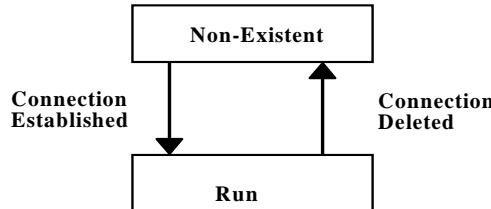
### **5-13.4 Object-specific Services**

The Presence Sensing Object provides no Object-specific services.

### 5-13.5 Behavior

The behavior of the Presence Sensing Object is illustrated in Figure 5-13.6, State Transition Diagram and Table 5-13.7, State Event Matrix. Table 5-13.8 indicates the accessibility of attributes.

**Figure 5-13.6 Presence Sensing Object State Transition Diagram**



**Table 5-13.7 Presence Sensing Object State Event Matrix**

Event	State	
	Non-Existent	Run
Connection established	Transition to Run	Ignore event
Connection deleted	Ignore event	Transition to Non-Existent
Get_Attribute_Single	Ignore event	Return value
Set_Attribute_Single	Ignore event	Accept value

**Table 5-13.8 Presence Sensing Object Attribute Access**

Attribute	State	
	Non-Existent	Running
Output	Not available	Read Only
Number of Attributes	Not available	Read Only
Attribute List	Not available	Read Only
Diagnostic	Not available	Read Only
On Delay	Not available	Read/Write
Off Delay	Not available	Read/Write
One Shot Delay	Not available	Read/Write
Operate Mode	Not available	Read/Write
Sensitivity	Not available	Read/Write
Target Margin	Not available	Read Only
Background Margin	Not available	Read Only
Min Detect Distance	Not available	Read/Write
Max Detect Distance	Not available	Read/Write
Detect Distance	Not available	Read Only

## 5-14 Parameter Object

### Class Code: 0F<sub>hex</sub>

Use of the Parameter Object provides a known, public interface to a device's configuration data. In addition, this object also provides all the information necessary to define and describe each of a device's individual configuration parameters.

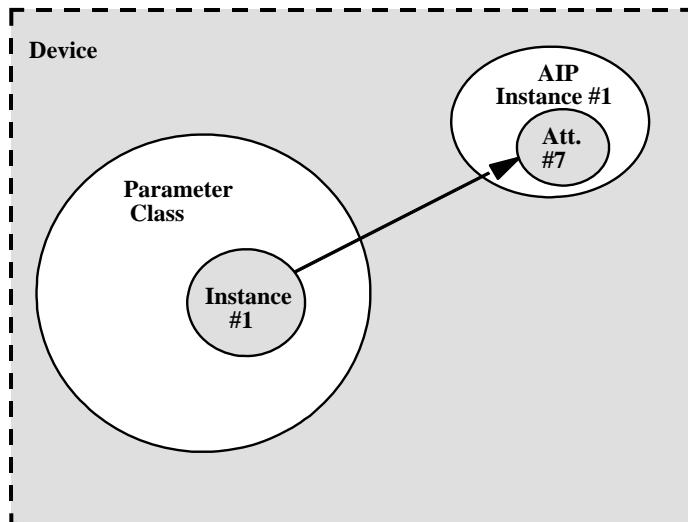
This object allows a device to fully identify a configurable parameter by supplying a full description of the parameter, including minimum and maximum values and a human-readable text string describing the parameter.

There must be one instance of this object class for each of the device's configurable parameters. The instances must start at instance one and increment by one with no gaps in the instances.

Each instance is linked to one of the configurable parameters, which is typically (but is not required to be) an attribute of one of the device's other objects. Changing the *Parameter Value* attribute of a Parameter Object causes a corresponding change in the attribute value indicated by the *Link Path* attribute (the attribute value being pointed to by the Parameter Object).

For example, Instance #1 of the Parameter Object below identifies the parameter represented by the Analog Input Point (AIP) Object, Instance #1, Attribute #7.

**Figure 5-14.1 Parameter Object representation of an Analog Input Point Attribute**



## 5-14.1 Class Attributes

Table 5-14.2 Parameter Object Class Attributes

Attrib ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1	This class attribute is optional and is described in Chapter 4 of this specification.					
2	Required	Get	Max Instance	UINT	Maximum instance number of an object currently created in this class level of the device.	The largest instance number of a created object at this class hierarchy level.
3 thru 7	These class attributes are optional and are described in Chapter 4 of this specification.					
8	Required	Get	Parameter Class Descriptor	WORD	Bits that describe parameters.	See Table 5-14.3
9	Required	Get	Configuration Assembly Instance	UINT	Instance number of the configuration assembly.	This attribute shall be set to zero if a configuration assembly is not supported.
10	Optional <sup>1</sup>	Set	Native Language	USINT	Language ID for all character array accesses.	0=English 1=French 2=Spanish 3=Italian 4=German 5=Japanese 6=Portuguese 7=Mandarin Chinese

1. If the Active Language attribute of the Identity Object (attribute ID 11) is supported, then this attribute shall not be supported.

### 5-14.1.1 Semantics

#### 5-14.1.1.1 Parameter Class Descriptor

The Parameter Class Descriptor attribute contains bits to describe parameter characteristics. The following bits are supported:

Table 5-14.3 Parameter Class Descriptor Bit Values

Bit	Definition
0	Supports Parameter Instances
1	Supports Full Attributes
2	Must do non-volatile storage save command
3	Params are stored in Non-Volatile storage

**5-14.1.1.1.1 Supports Parameter Instances, Bit 0**

“Supports Parameter Instances” indicates that Parameter Instances are present for each parameter. Parameter Instances contain all attributes or just the stub attributes. Bit values are:

**Table 5-14.4 “Supports Parameter Instance” Bit Values**

Value	Meaning
1	Individual Parameter instances ARE supported.
0	NO Parameter Instances are supported. Only configuration assembly instance used.

**5-14.1.1.1.2 Supports Full Attributes, Bit 1**

“Supports Full Attributes” indicates that each Parameter Instance contains all of the attributes and are not just stubs. Bit values are:

**Table 5-14.5 “Supports Full Attributes” Bit Values**

Value	Meaning
1	All Full Parameter Attributes ARE supported.
0	Only Parameter Instance stub attributes are supported.

**5-14.1.1.1.3 Must Do NV Storage Save Command, Bit 2**

“Must Do NV Storage Save Command” indicates that parameters are not stored into NV storage until a Save service is performed on the Parameter Object.

**Table 5-14.6 “Must Do VN Storage Save Command” Bit Values**

Value	Meaning
1	Must execute non-volatile storage save command. The <i>Save</i> service of the parameter class is used to save instance parameter values.
0	Do not have to execute non-volatile storage save command.

**5-14.1.1.1.4 Params Are Stored In Non-Volatile Storage, Bit 3**

“Params are stored in Non-Volatile Storage” indicates that parameters are stored in non-volatile storage. Bit values are:

**Table 5-14.7 “Params are stored in Non-Volatile Storage” Bit Values**

Value	Meaning
1	All Full parameters are stored in non-volatile storage.
0	Parameters are not stored in non-volatile storage.

A Parameter Object Stub is a shorthand version of a parameter object. The stub stores only the configuration data value, providing only a standard access point for the parameter.

## 5-14.2 Instance Attributes

Table 5-14.8 Parameter Object Instance Attributes

Attr ID	Need in Implementation	Access Rule	Stub/Full	Name	Data Type	Description of Attribute	Semantics of Values
1	Required	Set	Stub	Parameter Value	<i>data type</i> specified in Descriptor, Data Type and Data Size.	Actual value of parameter. It can be read from or written to. This attribute is read-only if bit 4 of Attribute #4 is TRUE.	
2	Required	Set	Stub	Link Path Size	USINT	Size of link path. If this attribute is 0, then no link is specified.	Number of bytes.
3	Required	Set	Stub	Link Path	Packed EPATH	CIP path to the object from where this parameter's value is retrieved.	The Link Path is limited to 255 bytes. See Appendix C.
4	Required	Get	Stub	Descriptor	WORD	Description of parameter.	See Table 5-14.9
5	Required	Get	Stub	Data Type	EPATH	Data type code.	See Appendix C-6.1
6	Required	Get	Stub	Data Size	USINT	Number of bytes in Parameter Value	
7	Conditional <sup>1</sup>	Get	Full	Parameter Name String	SHORT_STRING	A human-readable string representing the parameter name. For example, "Frequency #1"	The maximum number of characters is 16
8	Conditional <sup>1</sup>	Get	Full	Units String	SHORT_STRING	Engineering Unit String	The maximum number of characters is 4
9	Conditional <sup>1</sup>	Get	Full	Help String	SHORT_STRING	Help String	The maximum number of characters is 64
10	Conditional <sup>1</sup>	Get	Full	Minimum Value	<i>data type</i>	Generally, the minimum value to which the parameter can be set.	See semantics section 5-14.2.1.3
11	Conditional <sup>1</sup>	Get	Full	Maximum Value	<i>data type</i>	Generally, the maximum value to which the parameter can be set.	See semantics section 5-14.2.1.3
12	Conditional <sup>1</sup>	Get	Full	Default Value	<i>data type</i>	The actual value the parameter should be set to when the user wants the default for the parameter.	
13	Conditional <sup>1</sup>	Get	Full	Scaling Multiplier	UINT	Multiplier for Scaling Factor.	See Figure 5-14.11

Parameter Object, Class Code: 0F<sub>Hex</sub>

Attr ID	Need in Implementation	Access Rule	Stub/Full	Name	Data Type	Description of Attribute	Semantics of Values
14	Conditional <sup>1</sup>	Get	Full	Scaling Divisor	UINT	Divisor for Scaling Formula.	See Figure 5-14.11
15	Conditional <sup>1</sup>	Get	Full	Scaling Base	UINT	Base for Scaling Formula.	See Figure 5-14.11
16	Conditional <sup>1</sup>	Get	Full	Scaling Offset	INT (Can be negative)	Offset for Scaling Formula.	See Figure 5-14.11
17	Conditional <sup>1</sup>	Get	Full	Multiplier Link	UINT	Parameter Instance of Multiplier source.	See Table 5-14.16
18	Conditional <sup>1</sup>	Get	Full	Divisor Link	UINT	Parameter Instance of Divisor source.	See Table 5-14.16
19	Conditional <sup>1</sup>	Get	Full	Base Link	UINT	Parameter Instance of Base source.	See Table 5-14.16
20	Conditional <sup>1</sup>	Get	Full	Offset Link	UINT	Parameter Instance of Offset source.	See Table 5-14.16
21	Conditional <sup>1</sup>	Get	Full	Decimal Precision	USINT	Specifies number of decimal places to use when displaying the scaled engineering value. Also used to determine actual increment value so that incrementing a value causes a change in scaled engineering value to this precision.	
22	Optional	Get		International Parameter Name	STRINGI	Human readable string representing the parameter name	
23	Optional	Get		International Engineering Units	STRINGI	Engineering units associated with the parameter	
24	Optional	Get		International Help String	STRINGI	Help String	

<sup>1</sup> Required if the Parameter Object class descriptor bit 1 is set (supports full attributes). Not allowed if the Parameter Object class descriptor bit 1 is not set.

### 5-14.2.1 Semantics

#### 5-14.2.1.1 Descriptor Attribute

The Descriptor Instance Attribute contains these bits, which describe the parameter:

**Table 5-14.9 Semantics of Descriptor Instance Attribute**

Bit	Definition	Meaning
0	Supports Settable Path	Indicates that link path can be set.
1	Supports Enumerated Strings	Indicates that enumerated strings are supported and can be read with the Get_Enum_String service.
2	Supports Scaling	Indicates that the scaling factor should be implemented to present the value to the user in engineering units.
3	Supports Scaling Links	Indicates that the values for the scaling factor may be retrieved from other parameters. If the “Supports Scaling” descriptor bit is also set and the scaling link value is zero, the scaling value shall be used. If the “Supports Scaling” descriptor bit is also set and the scaling link value is not zero, the scaling link value shall be used.
4	Read Only Parameter	Indicates that the <b>Parameter Value</b> attribute can only be read, and not set.
5	Monitor Parameter	Indicates that the <b>Parameter Value</b> attribute is updated in real time by the device.
6	Supports Extended Precision Scaling	Indicates that the extended precision scaling factor should be implemented to present the value to the user in engineering units.
7	Supports non-consecutive enumerated strings	Obsolete.
8	Allows both enumeration and individual values	Obsolete.
9	Non-displayed parameter	The configuration tool shall not display this parameter
10	Indirect Parameter Reference	The parameter value represents the instance number of another parameter. The referenced parameter instance shall not be an additional indirect reference. A parameter value of 0 shall indicate no reference. The data type for this parameter shall be USINT, UINT or UDINT.
11	Not Addressable	The parameter is not directly addressable from the network by any of the Set_Attribute or Get_Attribute services.
12	Save Supported	This bit, when set, indicates that this parameter may be individually saved by sending the Save service to this parameter instance.
13	Apply Supported	This bit, when set, indicates the execution of the Apply service to this parameter instance is required before attribute changes affect instance behavior.
14	<b>Write Only Parameter</b>	Indicates that the <b>Parameter Value</b> attribute can only be set, and not read.
15	Reserved	This shall be set to zero

### 5-14.2.1.2 Data Type Attribute

The Data Type Instance Attribute specifies the data type of the parameter. This value shall be specified according to the format shown in Appendix C, Section C-6.1. The following data types are obsolete but may be encountered in older devices:

**Table 5-14.10 Obsolete Data Types Possible in the Parameter Object**

Attr Value	Definition	Description
1	WORD	16-bit word
2	UINT	16-bit unsigned integer
3	INT	16-bit signed integer
4	BOOL	Boolean
5	SINT	Short Integer
6	DINT	Double Integer
7	LINT	Long Integer
8	USINT	Unsigned Short Integer
9	UDINT	Unsigned Double Integer
10	ULINT	Unsigned Long Integer
11	REAL	Single floating point format (IEEE 754)
12	LREAL	Double floating point format (IEEE 754)
13	ITIME	Duration (short)
14	TIME	Duration
15	FTIME	Duration (high resolution)
16	LTIME	Duration (long)
17	DATE	Date (See Appendix J Volume I)
18	TIME_OF_DAY	Time of day (See Appendix J Volume I)
19	DATE_AND_TIME	Date and Time (See Appendix J Volume I)
20	STRING	8-bit per character string
21	STRING2	16-bit per character string
22	STRINGN	N-byte per character string
23	SHORT_STRING	Short N-byte character string
24	BYTE	8-bit string
25	DWORD	32-bit string
26	LWORD	64-bit string

Obsolete data type codes shall not be valid in devices developed to this specification.

## 5-14.2.1.3 Minimum and Maximum Value Attributes

Table 5-14.11 Minimum and Maximum Value Semantics

Data Type	Description and Semantics	Minimum Value Semantics	Maximum Value Semantics	Required/Optional/Not Allowed
BYTE	Bit String – 8 bit length	The most significant bit that is set in the minimum value is the lowest valid bit to be enumerated. The most significant bit that is set in the maximum value is the highest valid bit to be enumerated. A minimum value of zero means that bit 0 is the lowest valid bit to be enumerated. A maximum value of zero is not valid.  Example 1: Min value = 0b00001 Max value = 0b01000 Bits 0 – 3 are enumerated.  Example 2: Min value = 0b00001 Max value = 0b01011 Bits 0 – 3 are enumerated, bits 0 and 1 in max value are ignored.  Example 3: Min value = 0b00000 Max value = 0b01000 Bits 0 – 3 are enumerated, 0 min value implies bit 0.		Optional
WORD	Bit String – 16 bit length			
DWORD	Bit String – 32 bit length			
LWORD	Bit String – 64 bit length			
STRING <sup>1</sup>	String (2 byte length indicator, 1 byte per character)	Minimum string length	Maximum string length	Required
STRING2 <sup>1</sup>	String (2 byte length indicator, 2 bytes per character)	Minimum string length	Maximum string length	Required
STRINGN <sup>1</sup>	String (2 byte length indicator, N bytes per character)	Minimum string length	Maximum string length	Required
SHORT_STRING <sup>1</sup>	Character string (1 byte length indicator, 1 byte characters)	Minimum string length	Maximum string length	Required
STRINGI <sup>1</sup>	International Character String (see Appendix C for data type definition)	Minimum length (in characters) of the “stringcontents” component of the string’s implicit sequence	Maximum length (in characters) of the “stringcontents” component of the string’s implicit sequence	Required
EPATH <sup>1</sup>	Encoded Path	Minimum string length	Maximum string length	Optional
ENGUNIT	Engineering Units	The minimum value for this data type is not defined and shall not be specified in the EDS file or the parameter object instance.	The maximum value for this data type is not defined and shall not be specified in the EDS file or the parameter object instance.	Optional

**Parameter Object, Class Code: 0F<sub>Hex</sub>**

Data Type	Description and Semantics	Minimum Value Semantics	Maximum Value Semantics	Required/Optional/Not Allowed
All Other Data Types		The minimum numeric value that may be assigned to the data value.	The maximum numeric value that may be assigned to the data value.	Optional <sup>2</sup>

1 The STRING, STRING2, STRINGN, SHORT\_STRING, STRINGI and EPATH data types do not have a minimum or maximum value specification. The minimum and maximum value fields are used to present the minimum and maximum string or path lengths. In these cases, the Data Size parameter is used to represent the number of bytes required per character or encoding entry.

2 If the Minimum Value and/or Maximum Value is not specified then the minimum and/or maximum value for the parameter data value are as defined in CIP Common, Appendix C, Table C-2.1 based on the parameter data type.

**5-14.2.1.4 Scaling Factor Attributes**

The Scaling Factor represents actual UINT and INT parameter values in other formats. Use the following formula to determine the engineering value from the actual value:

**Figure 5-14.12 Determining Engineering Value from Actual Value**

$$\text{EngValue} = \frac{(\text{ActualValue} + \text{Offset}) * \text{Mult} * \text{Base}}{\text{Div}}$$

The engineering value can then be displayed to the user in the terms specified within the Decimal Precision Instance Attribute. Use the inverse of the Scaling Formula to determine the actual value from the engineering value:

**Figure 5-14.13 Determining Actual Value from Engineering Value**

$$\text{ActualValue} = \frac{(\text{EngValue} * \text{Div})}{\text{Mult} * \text{Base}} - \text{Offset}$$

The extended precision scaling factor adds extended precision to the standard scaling factor. Use this following formula to determine the engineering value from the actual value:

**Figure 5-14.14 Determining Engineering Value from Actual Value**

$$\text{EngValue} = \frac{(\text{ActualValue} + \text{Offset}) * \text{Mult} * \text{Base}}{\text{Div} * 10^{\text{Precision}}}$$

The engineering value can then be displayed to the user in the terms specified within the Decimal Precision attribute. Use the inverse of the extended precision scaling formula to determine the actual value from the engineering value:

**Figure 5-14.15 Determining Actual Value Using Precision and Scaling**

$$\text{ActualValue} = \frac{(\text{EngValue} * \text{Div} * 10^{\text{Precision}})}{\text{Mult} * \text{Base}} - \text{Offset}$$

For the scaling formula, the following attributes are required:

**Parameter Object, Class Code: 0F<sub>Hex</sub>****Table 5-14.16 Scaling Formula Attributes**

<b>Attribute</b>	<b>Meaning</b>
Multiplier Value	( <i>Mult</i> ) Multiplier value for the scaling formula. This value can be based on another parameter.
Divisor Value	( <i>Div</i> ) Divisor value for the scaling formula. This value can be based on another parameter specified in the Divisor Link.
Base Value	( <i>Base</i> ) Base value for the scaling formula. This value can be based on another parameter specified in the Base Link.
Offset Value	( <i>Offset</i> ) Offset value for the scaling formula. This value can be based on another parameter specified in the Offset Link. This attribute is an INT and can be negative.
Decimal Precision	(Precision) Precision value for the scaling formula. This value cannot be based on another parameter.

Scaling values (multiplier, divisor, base, and offset) can also be based on other parameters. Scaling Links specify from which instance that scaling value is to be retrieved. If the scaling link attribute is set to 0, then that scaling value is a constant and not linked to another parameter. The following attributes specify scaling links:

**Table 5-14.17 Scaling Links**

<b>Attribute</b>	<b>Meaning</b>
Multiplier Link	Specifies the parameter instance from where the Multiplier Value is retrieved.
Divisor Link	Specifies the parameter instance from where the Divisor Value is retrieved.
Base Link	Specifies the parameter instance from where the Base Value is retrieved.
Offset Link	Specifies the parameter instance from where the Offset Value is retrieved.

### 5-14.3 Common Services

The Parameter Object provides the following Common Services:

**Table 5-14.18 Parameter Object Common Services**

<b>Service Code</b>	<b>Need in Implementation</b>		<b>Service Name</b>	<b>Description of Service</b>
	<b>Class</b>	<b>Instance</b>		
0Dhex	Optional	Optional	Apply_Attributes	Causes parameter values whose use is <i>pending</i> to become actively used.
0Ehex	Required	Required	Get_Attribute_Single	Returns the contents of the specified attribute.
10hex	Optional	Required	Set_Attribute_Single	Modifies an attribute value.
01hex	Optional	Conditional <sup>4</sup>	Get_Attributes_All	Returns a predefined listing of this objects attributes (See the Get_Attributes_All Response definition below)
05hex	Optional	n/a	Reset	Resets all parameter values to the factory default.
15hex	Conditional <sup>2,3</sup>		Restore	Restores all parameter values from non-volatile storage.
		Conditional <sup>2</sup>		Restores parameter instance value from non-volatile storage.

**Parameter Object, Class Code: 0F<sub>Hex</sub>**

Service Code	Need in Implementation		Service Name	Description of Service
	Class	Instance		
16hex	Conditional <sup>2,3</sup>		Save	Saves all parameter values to non-volatile storage.
		Conditional <sup>2</sup>		Saves parameter instance value to non-volatile storage
18hex	n/a	Conditional <sup>1</sup>	Get_Member	Returns the content of a selected member of an attribute.

- 1) The Get\_Member service is required if any attributes with the International String (STRINGI) data type are implemented.  
 2) The Save and Restore services are optional. However, implementing either service requires implementation of both services.  
 3) If the Save and Restore services are implemented at the instance level, these services shall also be implemented at the class level.  
 4) Optional for Parameter Object Stubs, required for Full parameter Objects.

See the Appendix A for the definition of these common services.

**5-14.3.1 Get\_Attributes\_All Response**

At the Class level, the order of the attributes returned in the “Object/service specific reply data” portion of the Get\_Attributes\_All response is as follows:

**Table 5-14.19 Get\_Attributes\_All Response Service Data**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0								Revision (low byte) Default = 1
1								Revision (high byte) Default = 0
2								Max Instance (low byte)
3								Max Instance (high byte)
4								Parameter Class Descriptor (low byte)
5								Parameter Class Descriptor (high byte)
6								Configuration Assembly Instance (low byte)
7								Configuration Assembly Instance (high byte)
8								Native Language Default = 0

**Important:** Insert default values for all unsupported attributes.

At the Instance level, the setting of the Parameter Class Descriptor attribute dictates what attributes are returned by the Get\_Attributes\_All service:

**Table 5-14.20 Effect of Descriptor Attribute on Get\_Attributes\_All Response**

If the Parameter Class Descriptor Attribute:	Then:
does NOT have the “Supports Full Attributes” bit set,	only the implemented attributes marked <i>Stub</i> (attribute numbers 1–6) are returned by the Get_Attribute_All service
DOES have the “Supports Full Attributes” bit set,	The Get_Attribute_All service returns <i>all</i> implemented attributes (numbers 1–24)

**Parameter Object, Class Code: 0F<sub>Hex</sub>**

For Parameter object Stubs, the order of the attributes returned in the “Object/service specific reply data” portion of the Get\_Attributes\_All response is as follows:

**Table 5-14.21 Get\_Attributes\_All Response Service Data for Parameter Object Stub**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Parameter Value (low byte)							
n	Parameter Value (high byte)							
n+1	Link Path Size							
.	Link Path (1st byte)							
.	Link Path (last byte)							
.	Descriptor (low byte)							
.	Descriptor (high byte)							
.	Data Type							
.	Data Size							

For Full Parameter objects, the order of the attributes returned in the “Object/service specific reply data” portion of the Get\_Attributes\_All response is as follows:

**Table 5-14.22 Get\_Attributes\_All Response Service Data for Full Parameter Object**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Parameter Value (low byte)							
n	Parameter Value (high byte)							
n+1	Link Path Size							
.	Link Path (1st byte)							
.	Link Path (last byte)							
.	Descriptor (low byte)							
.	Descriptor (high byte)							
.	Data Type							
.	Data Size							
.	Parameter Name String (charcount) Default = 0							
.	Parameter Name String (1st byte)							
.	Parameter Name String (last byte)							
.	Units String (charcount) Default = 0							
.	Units String (1st byte)							
.	Units String (last byte)							
.	Help String (charcount) Default = 0							
.	Help String (1st byte)							
.	Help String (last byte)							
.	Minimum Value (low byte) Default = 0							
.	Minimum Value (high byte) Default = 0							
.	Maximum Value (low byte) Default = 0							

**Parameter Object, Class Code: 0F<sub>Hex</sub>**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
.								Maximum Value (high byte) Default = 0
.								Default Value (low byte) Default = 0
.								Default Value (high byte) Default = 0
.								Scaling Multiplier (low byte) Default = 1
.								Scaling Multiplier (high byte) Default = 0
.								Scaling Divisor (low byte) Default = 1
.								Scaling Divisor (high byte) Default = 0
.								Scaling Base (low byte) Default = 1
.								Scaling Base (high byte) Default = 0
.								Scaling Offset (low byte) Default = 0
.								Scaling Offset (high byte) Default = 0
.								Multiplier Link (low byte) Default = 0
.								Multiplier Link (high byte) Default = 0
.								Divisor Link (low byte) Default = 0
.								Divisor Link (high byte) Default = 0
.								Base Link (low byte) Default = 0
.								Base Link (high byte) Default = 0
.								Offset Link (low byte) Default = 0
.								Offset Link (high byte) Default = 0
.								Decimal Precision Default = 0
.								International Parameter Name [See note below]
.								International Engineering Units [See note below]
.								International Help String [See note below]

**Note:** For a device that has implemented the International String attributes, the Parameter Name character count, the Units String character count and the Help String character count shall be reported as zero. For device's that do not support the International String attributes, each (and every) International String attributes shall be reported as a single byte of zero, or may be missing from the response. If any International String attribute is supported, all three attributes shall be reported in the response.

**5-14.4****Object-specific Services**

The Parameter Object provides the following Object-specific service:

**Table 5-14.23 Parameter Object Specific Services**

Service Code	Need in Implementation		Service Name	Description of Service
4Bhex	n/a	Optional	Get_Enum_String	Use this service to read enumerated strings from the Parameter Instance. See below.

Enumerated strings are human-readable strings that describe either a *bit* or a *value*, depending on the parameter's data type.

**Parameter Object, Class Code: 0F<sub>Hex</sub>****Table 5-14.24 Enumerated Strings Type versus Parameter Data Type**

If the parameter's data type is:	Then the enumerated strings returned are:
BYTE, WORD, DWORD, LWORD	<i>Bit</i> enumerated strings.
USINT, SINT, UINT, INT, BOOL	<i>Value</i> enumerated strings.
All Other Data Types	Invalid for enumeration

If the parameter's data type is BYTE, WORD, DWORD or LWORD, requesting a Get\_Elem\_String service with an enumerated string number of 0 on a parameter returns a SHORT\_STRING that describes *bit* 0. Requesting a Get\_Elem\_String service with an enumerated string number of 1 on a parameter returns a SHORT\_STRING that describes *bit* 1. This continues up to the maximum bit number supported by the parameter.

If the parameter's data type is SINT, USINT, INT, UINT or BOOL, requesting a Get\_Elem\_String service with an enumerated string number of 0 on a parameter returns a SHORT\_STRING that describes the *value* of 0. Requesting a Get\_Elem\_String service with an enumerated string number of 1 returns a SHORT\_STRING that describes the *value* of 1.

The following parameters are defined for the Get\_Elem\_String service:

**Table 5-14.25 Parameters for Get\_Elem\_String Request**

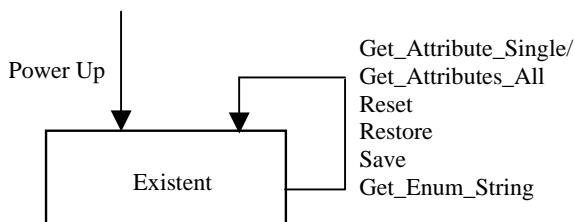
Name	Data Type	Description of Attribute	Semantics of Values
Enumerated String Number	USINT	Number of Enumerated String to retrieve.	Maximum possible range of values = 0 to 255. Value indicates bit offset (relative to zero) for bit enumerations, actual value (relative to zero) for value enumerations.

**Table 5-14.26 Parameters for Get\_Elem\_String Response**

Name	Data Type	Description of Attribute	Semantics of Values
Enumerated String	SHORT_STRING	Enumerated Strings	Maximum number of characters = 16

**5-14.5 Behavior**

The behavior of the Parameter Object is illustrated in the State Transition Diagram and State Event Matrix below.

**Figure 5-14.27 Parameter Object State Transition Diagram**

**Parameter Object, Class Code: 0F<sub>Hex</sub>**

**Table 5-14.28 Parameter Object State Event Matrix**

Event	State
	Existent
Get_Attribute_Single	Validates/service the request and returns the appropriate response.
Set_Attribute_Single	
Get_Attributes_All	
Reset	
Restore	
Save	
Get_Enum_String	

## 5-15 Parameter Group Object

**Class Code: 10 hex**

The Parameter Group Object identifies and provides access to groups of parameters in a device. Grouping parameters provides convenient access to related sets of parameters.

There must be one instance of this object class for each of the device's parameter groups. The instances must start at instance one and increment by one with no gaps in the instances.

Each Parameter Group Object contains a list of member Parameter Object Instances. Use the Get\_Attribute service to access these parameter object instances by number.

### 5-15.1 Class Attributes

**Table 5-15.1 Parameter Group Object Class Attribute**

Attribute ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1	Conditional	Get	Revision	UINT	Revision of this object	See below
2	Required	Get	Max Instance	UINT	Maximum instance number of an object currently created in this class level of the device.	The largest instance number of a created object at this class hierarchy level.
3 thru 7	These class attributes are optional and are described in Chapter 4 of this specification.					
8	Optional	Set	Native Language	USINT	Language ID for all STRING accesses.	0=English 1=French 2=Spanish 3=Italian 4=German 5=Japanese 6=Portuguese 7=Mandarin Chinese

#### 5-15.1.1 Semantics

##### 5-15.1.1.1 Revision – Class Attribute #1

The revision of the Parameter Group Object Class can be either 1 or 2, based upon the presence of the extended static attribute field.

If the value is 01, then this attribute is OPTIONAL in implementation and the abbreviated static attribute set layout is required to exist in all instances. If the value is greater than 01, then this attribute is REQUIRED and the extended static attribute set is defined to exist within all instances.

##### 5-15.1.1.2 Native Language – Class Attribute #8

The Native Language attribute is only used in revision 1 implementations of the Parameter Group. This attribute selects the language that will be presented in each instance's Group Name String attribute.

## 5-15.2 Instance Attributes, Abbreviated Static Set

This attribute layout only exists when the Parameter Group revision is one.

**Table 5-15.2 Parameter Group Object Instance Attributes, Abbreviated Static Set**

Attr ID	Need in Implem	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1	Required	Get	Group Name String	SHORT_STRING	A human-readable string representing the group name (e.g., Setup, Frequency Set)	Maximum number of characters = 16
2	Required	Get	Number of members in group	UINT	Number of parameters in group	
3	Required	Get	1st Parameter Number in Group	UINT	Parameter Instance Number	
4	Required	Get	2nd Parameter Number in Group	UINT	Parameter Instance Number	
...						
n	Required	Get	(n-2)th Parameter Number in Group	UINT	Parameter Instance Number	

The *Parameter Number* specifies the Parameter Instance of the nth member of a particular Parameter Group. Perform a Get\_Attribute service on this attribute to receive the Parameter Instance Number associated with the particular group member.

## 5-15.3 Instance Attributes, Extended Static Set

This attribute layout only exists when the Parameter Group revision is greater than one.

**Table 5-15.3 Parameter Group Object Instance Attributes, Extended Static Set**

Attr ID	Need in Implem	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1	Required	Get	Group Name String	SHORT_STRING	A human-readable string representing the group name (e.g., Setup, Frequency Set)	Maximum number of characters=16
2	Required	Get	Number of members in group	UINT	Number of parameters in group	
3	Optional	Get	International Group Name	STRING I	One or more international language strings representing the name of the parameter group	
4 – 15	Reserved					
16	Required	Get	1st Parameter Number in Group	UINT	Parameter Instance Number	
17	Required	Get	2nd Parameter Number in Group	UINT	Parameter Instance Number	
...						
n	Required	Get	(n-15)th Parameter Number in Group	UINT	Parameter Instance Number	

## 5-15.4 Common Services

The Parameter Group Object provides the following Common Services:

**Table 5-15.4 Parameter Group Object Common Services**

Service Code	Need in Implementation		Service Name	Service Description
	Class	Instance		
0Ehex	Required	Required	Get_Attribute_Single	Returns the contents of the specified attribute.
01hex	Optional	Optional	Get_Attributes_All	Returns a predefined listing of this objects attributes (See the Get_Attributes_All Response definition below)
10hex	Optional	n/a	Set_Attribute_Single	Modifies an attribute value.

See Appendix A for definitions of these common services.

### 5-15.4.1 Get Attributes\_All Response

At the Class level, the order of the attributes returned in the “Object/service specific reply data” portion of the Get\_Attributes\_All response is as follows:

**Table 5-15.5 Get\_Attributes\_All Response Data, Class Level**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0								Revision (low byte) Default = 1
1								Revision (high byte) Default = 0
2								Max Instance (low byte)
3								Max Instance (high byte)
4								Native Language Default = 0

**Abbreviated Static Set instance** - At the **Instance level**, the order of the attributes returned in the “Object/service specific reply data” portion of the Get\_Attributes\_All response for an Abbreviated Static Set instance is as follows:

**Table 5-15.6 Get\_Attributes\_All Response Data, Instance Level - Abbrev. Static Set**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0								Group Name (charcount)
1								Group Name (1st character)
n								Group Name (last character)
N+1								Number of members in group (low byte)
.								Number of members in group (high byte)
.								1st Parameter number in group (low byte)
.								1st Parameter number in group (high byte)
.								last Parameter number in group (low byte)
.								last Parameter number in group (high byte)

**Parameter Group Object, Class Code: 10<sub>Hex</sub>**

**Extended Static Set instance** - At the **Instance level**, the order of the attributes returned in the “Object/service specific reply data” portion of the Get\_Attributes\_All response for an Extended Static Set instance is as follows:

**Table 5-15.7 Get\_Attributes\_All Response Data, Instance Level - Extended Static Set**

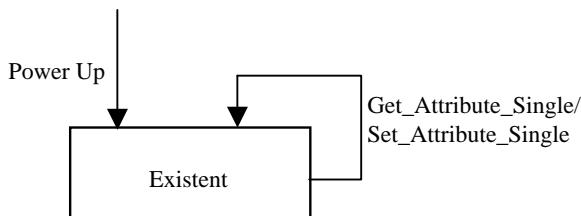
Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0								Group Name (International String)
1								Group Name (second octet)
n								Group Name (last octet)
n+1								Reserved – shall be reported as zero
n+2								Number of members in group (low byte)
.								Number of members in group (high byte)
.								1st Parameter number in group (low byte)
.								1st Parameter number in group (high byte)
.								last Parameter number in group (low byte)
.								last Parameter number in group (high byte)

**5-15.5 Object-specific Services**

The Parameter Group Object provides no Object-specific services on either the Class level or Instance level.

**5-15.6 Behavior**

The behavior of the Parameter Group Object is illustrated in the State Transition Diagram and State Event Matrix below.

**Figure 5-15.12 Parameter Group Object State Transition Diagram****Table 5-15.13 State Event Matrix for Parameter Group Object**

Event	State
	Existent
Get_Attribute_Single	Validates/services the request and returns the appropriate response.
Set_Attribute_Single	

## 5-16 Group Object

**Class Code: 12<sub>hex</sub>**

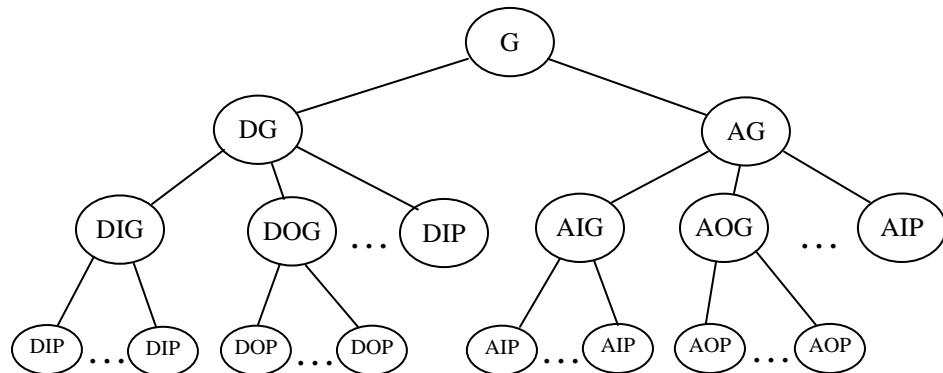
The Group Object binds other objects, typically discrete and analog, including AIP, AOP, AIG, AOG, DIP, DOP, DOG, or DIG. The objects bound to the Group must support the same attributes and services that apply to all of the bound objects (both inputs and outputs, discrete and analog) in a product.

You must establish the list of groups and/or points bound to the Group at power-up, as the Binding List is static and a Get-only attribute of the Group.

When to use the Group Object:

- when many attributes are shared among many analog and/or discrete input and output points and/or groups.
- to more efficiently access data by supporting services that may affect all members of the group.

**Figure 5-16.1 Group Object Application**



### 5-16.1 Class Attributes

**Table 5-16.2 Group Object Class Attributes**

Number	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1 thru 7	These class attributes are optional and are described in Chapter 4 of this specification.					

## 5-16.2 Instance Attributes

**Table 5-16.3 Group Object Instance Attributes**

Attr ID	Need in Implement	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1	Optional	Get	Number of Attributes	USINT	Number of attributes supported by the group	
2	Optional	Get	Attribute List	ARRAY of USINT	List of attributes supported	
3	Optional	Get	Number of Bound Instances	USINT	Number of points in a group	
4	Optional	Get	Binding	ARRAY of STRUCT:	List of instances in group	
			Class ID	UINT		
			Instance ID	UINT		
5	Optional	Get	Status	BOOL	Group is operating without alarms or faults	0=good; 1=alarm state
6	Optional	Get	Owner - Vendor ID	UINT	Vendor ID of group's owner	
7	Optional	Get	Owner - Serial Number	UDINT	32-bit serial number of group's owner	

## 5-16.3 Common Services

The Group Object provides the following Common Services:

**Table 5-16.4 Group Object Common Services**

Service Code	Need In Implementation		Service Name	Description of Service
	Class	Instance		
0E hex	Conditional *	Required	Get_Attribute_Single	Returns the contents of the specified attribute
01 hex	Optional	Optional	Get_Attributes_All	Returns a predefined listing of this object's attributes (See the Get_Attributes_All_Response definition below)

\* The Get\_Attribute\_Single service is REQUIRED if any attributes are implemented.

See Appendix A for definitions of these services.

### 5-16.3.1 Get\_Attributes\_All Response

At the **Class level**, the order of the attributes returned in the “Object/service specific reply data” portion (see Chapter 4 for a description of the Service Data field) of the Get\_Attributes\_All response is as follows:

**Group Object, Class Code: 12<sub>Hex</sub>****Table 5-16.5 Get\_Attributes\_All Service Data – Class Level**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0								Revision (low byte) Default = 1
1								Revision (high byte) Default = 0
2								Max Instance (low byte) Default = 0
3								Max Instance (high byte) Default = 0
4								Max ID Number of Class Attributes (low byte) Default = 0
5								Max ID Number of Class Attributes (high byte) Default = 0
6								Max ID Number of Instance Attributes (low byte) Default = 0
7								Max ID Number of Instance Attributes (high byte) Default = 0

**Important:** Insert default values for all unsupported attributes.

At the **Instance level**, the order of the attributes returned in the “Object/service specific reply data” portion of the Get\_Attributes\_All response is as follows:

**Table 5-16.6 Get\_Attributes\_All Service Data - Instance Level**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0								Number of Attributes Default = 0
1								Attribute List (attribute #1)
n								Attribute List (attribute #m)
n+1								Number of Bound Instances Default = 0
								Binding: Class ID #1 (low byte)
								Binding: Class ID #1 (high byte)
.								Binding: Instance ID #1 (low byte)
.								Binding: Instance ID #1 (high byte)
								Binding: Class ID #m (low byte)
								Binding: Class ID #m (high byte)
.								Binding: Instance ID #m (low byte)
.								Binding: Instance ID #m (high byte)
.	0	0	0	0	0	0	0	Status Default = 0
.								Owner Vendor ID (low byte) Default = 0
.								Owner Vendor ID (high byte) Default = 0
.								Owner Serial Number (low byte) Default = 0
.								Owner Serial Number Default = 0
.								Owner Serial Number Default = 0
.								Owner Serial Number (high byte) Default = 0

**Important:** Insert default values for all unsupported attributes.

**Important:** If the instance attribute ”Number of Attributes” is not supported, the default value of zero is to be inserted in its place and **no** members of the ”Attribute List” attribute will follow.

**Important:** If the instance attribute ”Number of Bound Instances” is not supported, the default value of zero is to be inserted into the response byte array and **no** Binding data will follow.

## 5-16.4 Object-specific Services

The Group Object provides no Object-specific services.

## 5-16.5 Behavior

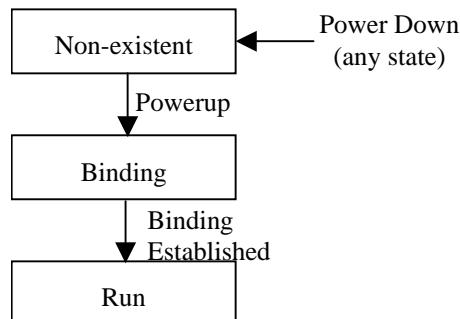
The primary purpose of the Group Object is to bind Analog and Discrete Groups and/or Points. An attribute of the Group will modify the behavior or report additional status information of all objects bound to the group.

The State Transition Diagram (STD) below illustrates the Behavior of a Group Object. The states shown are equivalent to the following:

**Table 5-16.7 Group Object State Definitions**

This state	Is equivalent to
Non-Existent	a module without power
	a module that has an unrecoverable fault (e.g. processor watchdog timeout)
	a major reconfiguration of the module (e.g. NVS Update)
Binding	when groups and/or points are added or bound to the Group (executed as part of Power Up cycle)
Run	the point at which Get and Set attribute services can be used to access the attributes of the Group Object

**Figure 5-16.8 Group Object State Transition Diagram**



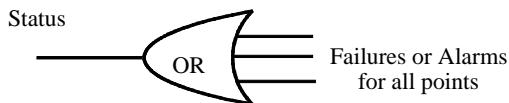
### 5-16.5.1 Attribute Access Rules

All attributes are gettable or settable according to their attribute access rules, but only in the Run state.

#### 5-16.5.1.1 Status Attribute

The *Status* attribute is simply a logical OR of all possible failure or alarm conditions for all points bound to the group.

**Figure 5-16.9 Group Object Status Attribute Operation**



#### 5-16.5.1.2 Owner Vendor ID, Owner Serial Number

These attributes may be implemented in both the point and group objects. If implemented in both, the values shall be the same for the point objects and the group object that binds the point object. If a connection to this object is made using the Connection Manager, then the value for the Owner Vendor Id and Owner Serial Number shall be the Originator Vendor Id and Originator Serial Number contained in the Forward\_Open request. These values shall be cleared when the connection is closed or deleted.

## 5-17 Discrete Input Group Object

**Class Code: 1D<sub>hex</sub>**

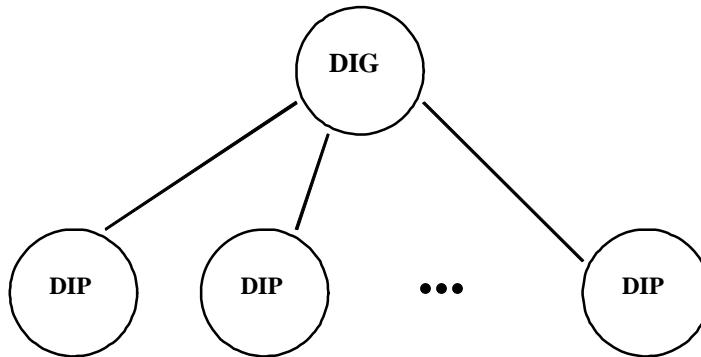
The Discrete Input Group (DIG) Object binds a group of discrete input points in a module. All points bound to the group share all attributes contained in the group. If an attribute is shared (points have the same attributes and the same attribute values) across more than one Discrete Input Point (DIP), then that attribute can be contained in a Discrete Input Group. A Discrete Input Point can be bound to more than one Discrete Input Group.

You must establish the list of discrete input points bound to the group at power-up, as the binding list is static and is a Get-only attribute of the group.

When to use the Discrete Input Group Object:

- when a single attribute is shared among many input points.
- to more efficiently access data (services that affect all members of a group can be supported by the DIG).

**Figure 5-17.1 Discrete Input Group Object Application**



### 5-17.1 Class Attributes

**Table 5-17.2 Discrete Input Group Object Class Attributes**

Number	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1 thru 7	These class attributes are optional and are described in Chapter 4 of this specification.					

## 5-17.2 Instance Attributes

**Table 5-17.3 Discrete Input Group Object Instance Attributes**

Attr ID	Need in Impl	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1	Optional	Get	Number of Attributes	USINT	Number of attributes supported by the device	
2	Optional	Get	Attribute List	ARRAY of USINT	List of attributes supported by the device	
3	Optional	Get	Number of Bound Instances	USINT	Number of points bound to this group	
4	Optional	Get	Binding	ARRAY of: UINT	List of all points bound to this group Instance ID	Class DIP Instance #X Class DIP Instance #Y...
5	Optional	Get	Status	BOOL	Status for all discrete input points in the group	0 = OK 1 = Product specific alarm or status
6	Optional	Set	Off_On Delay	UINT	filter time for off to on transition 0 - 65,535 microseconds <sup>1</sup>	The default value is 0.
7	Optional	Set	On_Off Delay	UINT	filter time for on to off transition 0 - 65,535 microseconds <sup>2</sup>	The default value is 0.

<sup>1</sup> The input must be on for the amount of filter time specified by the OFF\_ON DELAY attribute before the ON state is recorded in the VALUE attribute.

<sup>2</sup> The input must be off for the amount of filter time specified by the ON\_OFF DELAY attribute before the OFF state is recorded in the VALUE attribute.

## 5-17.3 Semantics

### 5-17.3.1 On\_OffDelay, Off\_OnDelay

When any of these attributes are implemented at the Group level, the corresponding attribute of the point objects bound to the group shall assume the value of the Group level object attribute. The attribute shall be implemented at the point level and shall be “Get” only.

## 5-17.4 Common Services

The Discrete Input Group Object provides the following Common Services:

Discrete Input Group Object, Class Code: 1D<sub>Hex</sub>

Table 5-17.4 Discrete Input Group Object Common Services

Service Code	Need In Implementation		Service Name	Description of Service
	Class	Instance		
0E hex	Conditional*	Required	Get_Attribute_Single	Returns the contents of the specified attribute
10 hex	N/A	Optional	Set_Attribute_Single	Modifies an attribute value
01 hex	Optional	Optional	Get_Attributes_All	Returns a predefined listing of this object's attributes (See the Get_Attributes_All_Response definition below)
02 hex	N/A	Optional	Set_Attributes_All	Modifies the contents of the attributes of the class or object.

\* The Get\_Attribute\_Single service is REQUIRED if any attributes are implemented.

See Appendix A for definitions of these services.

5-17.4.1 **Get\_Attributes\_All Response**

At the **Class level**, the order of the attributes returned in the “Object/service specific reply data” portion of the Get\_Attributes\_All response is as follows:

Table 5-17.5 Get\_Attributes\_All Response Data – Class Level

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0								Revision (low byte) Default = 1
1								Revision (high byte) Default = 0
2								Max Instance (low byte) Default = 0
3								Max Instance (high byte) Default = 0
4								Number of Instances (low byte) Default = 0
5								Number of Instances (high byte) Default = 0
6								Max ID Number of Class Attributes (low byte) Default = 0
7								Max ID Number of Class Attributes (high byte) Default = 0
8								Max ID Number of Instance Attributes (low byte) Default = 0
9								Max ID Number of Instance Attributes (high byte) Default = 0

**Important:** Insert default values for all unsupported attributes.

At the **Instance level**, the order of the attributes returned in the “Object/service specific reply data” portion of the Get\_Attributes\_All response is as follows:

Table 5-17.6 Get\_Attributes\_All Response Data – Instance Level

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0								Number of Attributes Default = 0
1								Attribute List (attribute #1)
n								Attribute List (attribute #m)
n+1								Number of Bound Instances Default = 0
.								Binding : Instance ID #1 (low byte)
.								Binding : Instance ID #1 (high byte)
.								Binding : Instance ID #m (low byte)
.								Binding : Instance ID #m (high byte)

**Discrete Input Group Object, Class Code: 1D<sub>Hex</sub>**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
.	0	0	0	0	0	0	0	Status Default = 0
.	Off_On Delay (low byte) Default = 0							
.	Off_On Delay (high byte) Default = 0							
.	On_Off Delay (low byte) Default = 0							
.	On_Off Delay (high byte) Default = 0							

**Important:** Insert default values for all unsupported attributes.

**Important:** If the instance attribute "Number of Attributes" is not supported, the default value of zero is to be inserted in its place and **no** members of the "Attribute List" attribute will follow.

**Important:** If the instance attribute "Number of Bound Instances" is not supported, the default value of zero is to be inserted into the response byte array and **no** Binding data will follow.

#### 5-17.4.2 Set\_Attributes\_All Request

No settable attributes currently exist at the **Class level** for the Discrete Input Group Object.

At the **Instance level**, the order of attributes passed in the "Object/service specific request data" portion of the Set\_Attributes\_All request is as follows:

**Table 5-17.7 Set\_Attributes\_All Request Service Data – Instance Level**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Off_On Delay (low byte) Default = 0							
1	Off_On Delay (high byte) Default = 0							
2	On_Off Delay (low byte) Default = 0							
3	On_Off Delay (high byte) Default = 0							

**Important:** The Set\_Attributes\_All service is to be supported only if **all** settable attributes shown above are implemented as settable.

#### 5-17.5 Object-specific Services

The Discrete Input Group Object does not support any Object-specific services.

#### 5-17.6 Behavior

The primary purpose of the DIG is to bind discrete input points.

An attribute of the DIG will modify the behavior or report additional status information of all DIPs bound to the group.

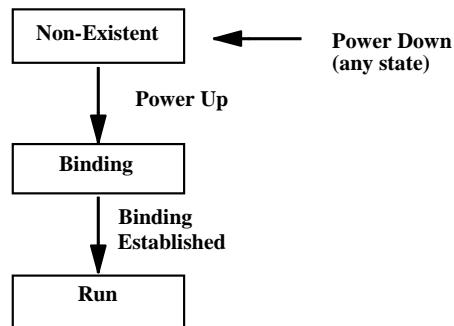
The State Transition Diagram below provides a graphical description of the events and corresponding state transitions. A subset of the states and events may be supported in an application, but the behavior must be consistent.

**Discrete Input Group Object, Class Code: 1D<sub>Hex</sub>**

The states shown are equivalent to the following:

**Table 5-17.8 Discrete Input Group Object State Definitions**

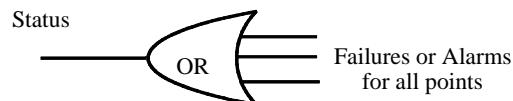
<b>This state</b>	<b>Is equivalent to</b>
Non-Existent	a module without power
	a module that has an unrecoverable fault (e.g. processor watchdog timeout)
	a major reconfiguration of the module (e.g. NVS Update)
Binding	when discrete input points are added or bound to the Discrete Input Group (executed as part of Power Up cycle)
Run	the point at which Get and Set attribute services can be used to access the attributes of the Group Object

**Figure 5-17.9 State Transition Diagram for Discrete Input Group Object****5-17.6.1 Attribute Access Rules**

All attributes are gettable or settable according to their attribute access rules, but only in the Run state.

**5-17.6.1.1 Status Attribute**

The *Status* attribute is simply a logical OR of all possible failure or alarm conditions for all points bound to the group.

**Figure 5-17.10 Discrete Input Group Object Status Attribute Operation**

## 5-18 Discrete Output Group Object

**Class Code: 1E<sub>hex</sub>**

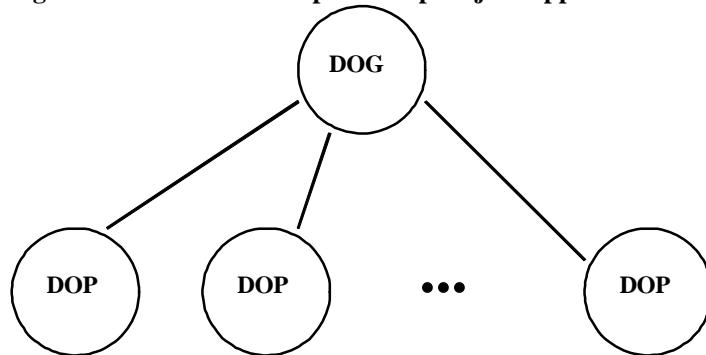
The Discrete Output Group (DOG) Object binds a group of discrete output points in a module. All points bound to the group share all attributes contained in the group. If an attribute is shared across more than one Discrete Output Point (DOP), then it can be contained in a Discrete Output Group. A Discrete Output Point can also be bound to more than one Discrete Output Group.

You must establish the list of discrete output points bound to the group at power-up, as the binding list is static and is a Get-only attribute of the group.

When to use the Discrete Output Group Object:

- when a single attribute is shared among many output points.
- to more efficiently access data (services that affect all members of a group can be supported by the DOG).

**Figure 5-18.1 Discrete Output Group Object Application**



### 5-18.1 Class Attributes

**Table 5-18.2 Discrete Output Group Object Class Attributes**

Attribute ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1 thru 7	These class attributes are optional and are described in Chapter 4 of this specification.					

## 5-18.2 Instance Attributes

**Table 5-18.3 Discrete Output Group Object Instance Attributes**

Attr ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1	Optional	Get	Number of Attributes	USINT	Number of attributes supported by the device	
2	Optional	Get	Attribute List	ARRAY of USINT	List of attributes supported by the device	
3	Optional	Get	Number of Bound Instances	USINT	Number of points bound to this group	
4	Optional	Get	Binding	ARRAY of:	List of all points bound to this group	
			Instance ID	UINT		
5	Optional	Get	Status	BOOL	Status for all discrete output points in the group	0 = OK 1 = Product specific alarm or status
6	Required	Set	Command	BOOL	Changes state of all DOPs in group to Idle Mode or Run Mode	0=idle 1=run
7	Optional	Set	Fault Action	BOOL	State of output after recoverable failure	0=Fault Value attribute; 1=hold last state
8	Optional	Set	Fault Value	BOOL	User-defined value for use with Fault State attribute	0=off; 1=on
9	Optional	Set	Idle Action	BOOL	State of output during idle	0=Idle Value attribute; 1=hold last state
10	Optional	Set	Idle Value	BOOL	User-defined value for use with Idle State attribute	0=off; 1=on

## 5-18.3 Semantics

### 5-18.3.1 Command, Fault Action, Fault Value, Idle Action, Idle Value

When any of these attributes are implemented at the Group level, the corresponding attribute of the point objects bound to the group shall assume the value of the Group level object attribute. The attribute shall be implemented at the point level and shall be “Get” only.

The Idle Action and Idle Value behavior shall be the power up default behavior and the behavior when in the available state.

## 5-18.4 Common Services

The Discrete Output Group Object provides the following Common Services:

**Table 5-18.4 Discrete Output Group Object Common Services**

Service Code	Need In Implementation		Service Name	Description of Service
	Class	Instance		
0E hex	Conditional *	Required	Get_Attribute_Single	Returns the contents of the specified attribute
10 hex	N/A	Required	Set_Attribute_Single	Modifies an attribute value
01 hex	Optional	Optional	Get_Attributes_All	Returns a predefined listing of this object's attributes (See the Get_Attributes_All_Response definition below)
02 hex	N/A	Optional	Set_Attributes_All	Modifies the contents of the attributes of the class or object.

\* The Get\_Attribute\_Single service is REQUIRED if any attributes are implemented.

See Appendix A for definitions of these services.

### 5-18.4.1 Get\_Attributes\_All Response

At the **Class level**, the order of the attributes returned in the “Object/service specific reply data” portion of the Get\_Attributes\_All response is as follows:

**Table 5-18.5 Get\_Attributes All Response Data – Class Level**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0								Revision (low byte) Default = 1
1								Revision (high byte) Default = 0
2								Max Instance (low byte) Default = 0
3								Max Instance (high byte) Default = 0
6								Max ID Number of Class Attributes (low byte) Default = 0
7								Max ID Number of Class Attributes (high byte) Default = 0
8								Max ID Number of Instance Attributes (low byte) Default = 0
9								Max ID Number of Instance Attributes (high byte) Default = 0

**Important:** Insert default values for all unsupported attributes.

Discrete Output Group Object, Class Code: 1E<sub>Hex</sub>

At the **Instance level**, the order of the attributes returned in the “Object/service specific reply data” portion of the Get\_Attributes\_All response is as follows:

**Table 5-18.6 Get\_Attributes All Response Data – Instance Level**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Number of Attributes Default = 0							
1	Attribute List (attribute #1)							
n	Attribute List (attribute #m)							
n+1	Number of Bound Instances Default = 0							
.	Binding : Instance ID #1 (low byte)							
.	Binding : Instance ID #1 (high byte)							
.	Binding : Instance ID #m (low byte)							
.	Binding : Instance ID #m (high byte)							
.	0	0	0	0	0	0	0	Status Default = 0
.	0	0	0	0	0	0	0	Command
.	0	0	0	0	0	0	0	Fault State Default = 0
.	0	0	0	0	0	0	0	Fault Value Default = 0
.	0	0	0	0	0	0	0	Idle State Default = 0
.	0	0	0	0	0	0	0	Idle Value Default = 0

**Important:** Insert default values for all unsupported attributes.

**Important:** If the instance attribute ”Number of Attributes” is not supported, the default value of zero is to be inserted in its place and **no** members of the ”Attribute List” attribute will follow.

**Important:** If the instance attribute ”Number of Bound Instances” is not supported, the default value of zero is to be inserted into the response byte array and **no** Binding data will follow.

#### 5-18.4.2 Set\_Attributes\_All Request

No settable attributes currently exist at the **Class level** for the Discrete Output Group Object.

At the **Instance level**, the order of attributes passed in the “Object/service specific request data” portion (see Chapter 4-5.2) is as follows:

**Table 5-18.7 Set\_Attributes All Request Data – Instance Level**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	0	0	0	Fault State
1	0	0	0	0	0	0	0	Fault Value
2	0	0	0	0	0	0	0	Idle State
3	0	0	0	0	0	0	0	Idle Value

**Important:** The Set\_Attributes\_All service is to be supported only if **all** settable attributes shown above are implemented as settable.

### 5-18.5 Object-specific Services

The Discrete Output Group Object provides no Object-specific services.

### 5-18.6 Behavior

The primary purpose of the Discrete Output Group Object is to bind discrete output points.

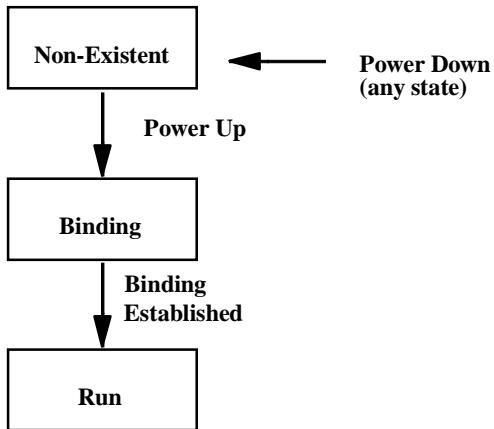
An attribute of the DOG will modify the behavior or report additional status information of all DOPs bound to the group.

The State Transition Diagram below provides a graphical description of the events and corresponding state transitions. A subset of the states and events may be supported in an application, but the behavior must be consistent.

The states shown are equivalent to the following:

**Table 5-18.8 Discrete Output Group Object State Definitions**

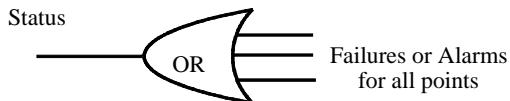
This state	Is equivalent to
Non-Existent	a module without power
	a module that has an unrecoverable fault (e.g. processor watchdog timeout)
	a major reconfiguration of the module (e.g. NVS Update)
Binding	when discrete output points are added or bound to the Discrete Output Group (executed as part of Power Up cycle)
Run	the point at which Get and Set attribute services can be used to access the attributes of the Group Object

**Table 5-18.9 Discrete Output Group Object State Transition Diagram****5-18.6.1 Attribute Access Rules**

All attributes are gettable or settable according to their attribute access rules, but only in the Run state.

**5-18.6.1.1 Status Attribute**

The *Status* attribute is simply a logical OR of all possible failure or alarm conditions for all points bound to the group.

**Figure 5-18.10 Discrete Output Group Object Status Attribute Operation**

## 5-19 Discrete Group Object

**Class Code: 1F<sub>hex</sub>**

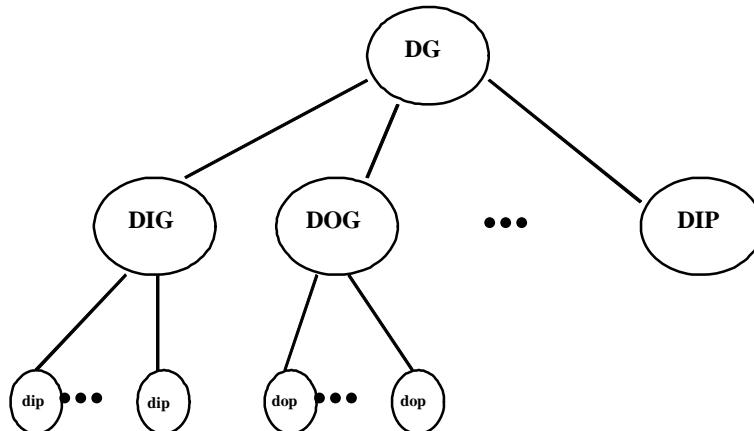
The Discrete Group (DG) Object binds other discrete objects including DIP, DOP, DIG or DOG. The objects (both input and output) bound to the Discrete Group must support the same attributes and services.

You must establish the list of discrete groups and/or points bound to the Discrete Group at power-up, as the Binding List is static and a Get-only attribute of the Discrete Group.

When to use the Discrete Group Object:

- when a single attribute is shared among many discrete input and output points and/or groups.
- to more efficiently access data (services that affect all members of the group can be supported by the DG).

**Figure 5-19.1 Discrete Group Object Application**



For example:

The Discrete Group Object could be used when a module contains 4 DIPs and 4 DOPs; the DIPs are grouped into a DIG to share a common attribute, and the DOPs are grouped into a DOG to share a common output behavior.

You want all points to share a common status. The DG has the common attribute Status and binds the DIG and DOG, which in turn bind the individual points.

### 5-19.1 Class Attributes

**Table 5-19.2 Discrete Group Object Class Attributes**

Attribute ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1 thru 7	These class attributes are optional and are described in Chapter 4 of this specification.					

## 5-19.2 Instance Attributes

**Table 5-19.3 Discrete Group Object Instance Attributes**

Attr ID	Need in Impl	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1	Optional	Get	Number of Attributes	USINT	Number of attributes supported by the group	
2	Optional	Get	Attribute List	ARRAY of USINT	List of attributes supported	
3	Optional	Get	Number of Bound Instances	USINT	Number of points in a group	
4	Optional	Get	Binding	ARRAY of STRUCT:	List of all instances group	
			Class ID	UINT		
			Instance ID	UINT		
5	Optional	Get	Status	BOOL	Group is operating without alarms or faults	0=good; 1=alarm state

## 5-19.3 Common Services

The Discrete Group Object provides the following Common Services:

**Table 5-19.4 Discrete Group Object Common Services**

Service Code	Need In Implementation		Service Name	Description of Service
	Class	Instance		
0E hex	Conditional *	Required	Get_Attribute_Single	Returns the contents of the specified attribute
01 hex	Optional	Optional	Get_Attributes_All	Returns a predefined listing of this object's attributes (See the Get_Attributes_All_Response definition below)

\* The Get\_Attribute\_Single service is REQUIRED if any attributes are implemented.

See Appendix A for definitions of these services.

### 5-19.3.1 Get\_Attributes\_All Response

At the **Class level**, the order of the attributes returned in the “Object/service specific reply data” portion of the Get\_Attributes\_All response is as follows:

**Table 5-19.5 Get\_Attributes All Response Data – Class Level**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0								Revision (low byte) Default = 1
1								Revision (high byte) Default = 0
2								Max Instance (low byte) Default = 0
3								Max Instance (high byte) Default = 0
6								Max ID Number of Class Attributes (low byte) Default = 0
7								Max ID Number of Class Attributes (high byte) Default = 0
8								Max ID Number of Instance Attributes (low byte) Default = 0
9								Max ID Number of Instance Attributes (high byte) Default = 0

**Important:** Insert default values for all unsupported attributes.

At the **Instance level**, the order of the attributes returned in the “Object/service specific reply data” portion of the Get\_Attributes\_All response is as follows:

**Table 5-19.6 Get\_Attributes All Response Data – Instance Level**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0								Number of Attributes Default = 0
1								Attribute List (attribute #1)
n								Attribute List (attribute #m)
n+1								Number of Bound Instances Default = 0
.								Binding : Class ID #1 (low byte)
.								Binding : Class ID #1 (high byte)
.								Binding : Instance ID #1 (low byte)
.								Binding : Instance ID #1 (high byte)
.								Binding : Class ID #m (low byte)
.								Binding : Class ID #m (high byte)
.								Binding : Instance ID #m (low byte)
.								Binding : Instance ID #m (high byte)
.	0	0	0	0	0	0	0	Status Default = 0

**Important:** Insert default values for all unsupported attributes.

**Important:** If the instance attribute ”Number of Attributes” is not supported, the default value of zero is to be inserted in its place and **no** members of the ”Attribute List” attribute will follow.

**Important:** If the instance attribute ”Number of Bound Instances” is not supported, the default value of zero is to be inserted into the response byte array and **no** Binding data will follow.

## 5-19.4 Object-specific Services

The Discrete Group Object provides no Object-specific services

## 5-19.5 Behavior

The primary purpose of the Discrete Group is to bind Discrete Groups and/or Points.

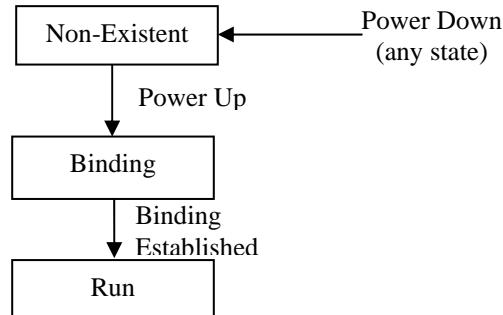
An attribute of the DG modifies the behavior or reports additional status information of all objects bound to the group.

The State Transition Diagram below illustrates the behavior of a Discrete Group.

**Table 5-19.7 Discrete Group Object State Definitions**

This state	Is equivalent to
Non-Existent	a module without power
	a module that has an unrecoverable fault (e.g. processor watchdog timeout)
	a major reconfiguration of the module (e.g. NVS Update)
Binding	when the discrete groups and/or points are added or bound to the Discrete Group (executed as part of Power Up cycle)
Run	the point at which Get and Set attribute services can be used to access the attributes of the Discrete Group Object

**Figure 5-19.8 State Transition Diagram for Discrete Group Object**



### 5-19.5.1 Attribute Access Rules

All attributes are gettable or settable according to their attribute access rules, but only in the Run state.

#### 5-19.5.1.1 Status Attribute

The *Status* attribute is simply a logical OR of all possible failure or alarm conditions for all points bound to the group.

**Figure 5-19.9 Discrete Group Object Status Attribute Function**



## 5-20 Analog Input Group Object

**Class Code: 20<sub>hex</sub>**

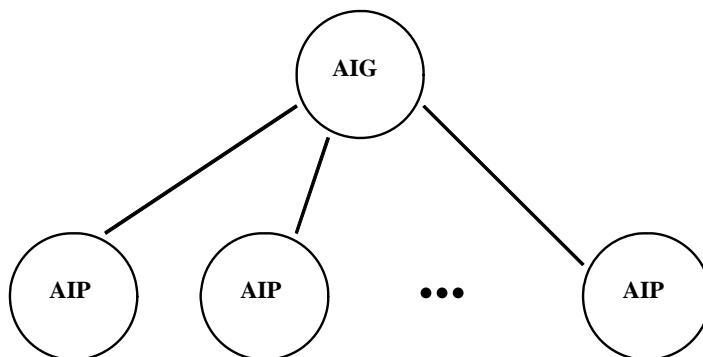
The Analog Input Group Object (AIG) binds a group of analog input points in a module. All points bound to the group share all attributes contained in the group . If an attribute is shared (points have the same attributes AND the same attribute values) across more than one Analog Input Point (AIP), then that attribute can be contained in an Analog Input Group. An Analog Input Point can be bound to more than one Analog Input Group.

You must establish the list of analog input points bound to the group at power-up, as the binding list is static and is a Get-only attribute of the group.

When to use the Analog Input Group Object:

- when a single attribute is shared among many input points.
- to more efficiently access data (services that affect all members of the group can be supported by the AIG).

**Figure 5-20.1 Analog Input Group Object Application**



### 5-20.1 Class Attributes

**Table 5-20.2 Analog Input Group Object Class Attributes**

Attribute ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1 thru 7	These class attributes are optional and are described in Chapter 4 of this specification.					

## 5-20.2 Instance Attributes

**Table 5-20.3 Analog Input Group Object Instance Attributes**

Attr ID	Need in Implem	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1	Optional	Get	Number of Attributes	USINT	Number of attributes supported by the group	
2	Optional	Get	Attribute List	ARRAY of USINT	List of attributes supported	
3	Optional	Get	Number of Bound Instances	USINT	Number of points in a group	
4	Optional	Get	Bound Instances	ARRAY of UINT	List of all points bound to this group	
5	Optional	Get	Status	BOOL	Group is operating without alarms or faults	0=good; 1=alarm state
6	Optional	Get	Owner - Vendor ID	UINT	Vendor ID of group's owner	
7	Optional	Get	Owner - Serial Number	UDINT	32-bit serial number of group's owner	
8	Optional	Set	Value Data Type	USINT	Determines the data type of AIP's Value	0 = INT 1 = REAL 2 = USINT 3 = SINT 4 = DINT 5 = LINT 6 = UINT 7 = UDINT 8 = ULINT 9 = LREAL 100=vendor specific
9	Reserved for CIP					
10	Optional	Set	Temp Mode	BOOL	Temperature scale to use when reporting a temperature value	0 = Celsius 1 = Fahrenheit

## 5-20.3 Semantics

### 5-20.3.1 Value Data Type

When this attribute is implemented at the Group level, the corresponding attribute of the point objects bound to the group shall assume the value of the Group level object attribute.

## 5-20.4 Common Services

The Analog Input Group Object provides the following Common Services:

**Table 5-20.4 Analog Input Group Object Common Services**

Service Code	Need In Implementation		Service Name	Description of Service
	Class	Instance		
0E hex	Conditional <sup>1</sup>	Required	Get_Attribute_Single	Returns the contents of the specified attribute
10 hex	N/A	Required <sup>2</sup>	Set_Attribute_Single	Modifies an attribute value
01 hex	Optional	Optional	Get_Attributes_All	Returns a predefined listing of this object's attributes (See the Get_Attributes_All_Response definition below)
02 hex	N/A	Optional	Set_Attributes_All	Modifies the contents of the attributes of the class or object.

<sup>1</sup> The Get\_Attribute\_Single service is REQUIRED if any attributes are implemented.

<sup>2</sup> The Set\_Attribute\_Single service is required only if Instance Attributes #8 and/or #10 are implemented.

See Appendix A for definitions of these services.

### 5-20.4.1 Get\_Attributes\_All Response

At the **Class level**, the order of the attributes returned in the “Object/service specific reply data” portion of the Get\_Attributes\_All response is as follows:

**Table 5-20.5 Get\_Attributes All Response Data – Class Level**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0								Revision (low byte) Default = 1
1								Revision (high byte) Default = 0
2								Max Instance (low byte) Default = 0
3								Max Instance (high byte) Default = 0
4								Max ID Number of Class Attributes (low byte) Default = 0
5								Max ID Number of Class Attributes (high byte) Default = 0
6								Max ID Number of Instance Attributes (low byte) Default = 0
7								Max ID Number of Instance Attributes (high byte) Default = 0

**Important:** Insert default values for all unsupported attributes.

**Analog Input Group Object, Class Code: 20<sub>Hex</sub>**

At the **Instance level**, the order of the attributes returned in the “Object/service specific reply data” portion of the Get\_Attributes\_All response is as follows:

**Table 5-20.6 Get\_Attributes All Response Data – Instance Level**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Number of Attributes Default = 0							
1	Attribute List (attribute #1)							
n	Attribute List (attribute #m)							
n+1	Number of Bound Instances Default = 0							
.	Binding : Instance ID #1 (low byte)							
.	Binding : Instance ID #1 (high byte)							
.	Binding : Instance ID #m (low byte)							
.	Binding : Instance ID #m (high byte)							
.	0	0	0	0	0	0	0	Status Default = 0
.	Owner Vendor ID (low byte) Default = 0							
.	Owner Vendor ID (high byte) Default = 0							
.	Owner Serial Number (low byte) Default = 0							
.	Owner Serial Number Default = 0							
.	Owner Serial Number Default = 0							
.	Owner Serial Number (high byte) Default = 0							
.	Value Data Type Default = 0							
.	0	0	0	0	0	0	0	Temp Mode Default = 0

**Important:** Insert default values for all unsupported attributes.

**Important:** If the instance attribute ”Number of Attributes” is not supported, the default value of zero is to be inserted in its place and **no** members of the ”Attribute List” attribute will follow.

**Important:** If the instance attribute ”Number of Bound Instances” is not supported, the default value of zero is to be inserted into the response byte array and **no** Binding data will follow.

#### 5-20.4.2 Set\_Attributes\_All Request

No settable attributes currently exist at the **Class level** for the Analog Input Group Object.

At the **Instance level**, the order of attributes passed in the “Object/service specific request data” portion of the Set\_Attributes\_All request is as follows:

**Table 5-20.7 Set\_Attributes\_All Request Service Data – Instance Level**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Value Data Type							
1	0	0	0	0	0	0	0	Temp Mode

**Important:** The Set\_Attributes\_All service is to be supported only if **all** settable attributes shown above are implemented as settable.

## 5-20.5 Object-specific Services

The Analog Input Group Object provides no Object-specific services.

## 5-20.6 Behavior

The primary purpose of the Analog Input Group Object is to bind analog input points.

An attribute of the AIG modifies the behavior or reports additional status information of all AIPs bound to the group.

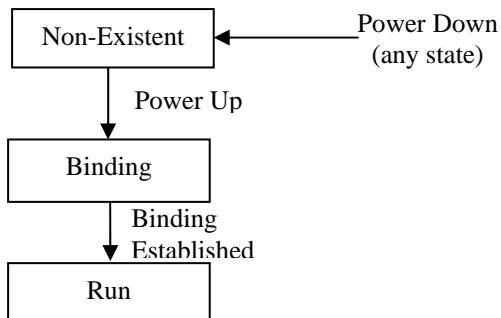
The State Transition Diagram below provides a graphical description of the events and corresponding state transitions. A subset of the states and events may be supported in an application, but the behavior must be consistent.

The states shown are equivalent to the following:

**Table 5-20.8 Analog Input Group Object State Definitions**

This state	Is equivalent to
Non-Existent	a module without power
	a module that has an unrecoverable fault (e.g. processor watchdog timeout)
	a major reconfiguration of the module (e.g. NVS Update)
Binding	when analog input points are added or bound to the Analog Input Group (executed as part of Power Up cycle)
Run	the point at which Get and Set attribute services can be used to access the attributes of the Group Object

**Figure 5-20.9 State Transition Diagram for Analog Input Group Object**



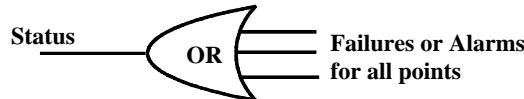
### 5-20.6.1 Attribute Access Rules

All attributes are gettable or settable according to their attribute access rules.

#### 5-20.6.1.1 Status Attribute

The *Status* attribute is simply a logical OR of all possible failure or alarm conditions for all points bound to the group.

**Figure 5-20.10 Analog Input Group Object Status Attribute Function**



#### 5-20.6.1.2 Value Data Type Attribute

The *Value Data Type* attribute determines the data type to be used by the attribute Value (and perhaps other attributes as well). If Value Data Type is not used, then Value defaults to INT. Value (and other attributes) may behave as REAL, USINT, or any other data type based upon the definition of Value Data Type, which ultimately provides the ability for the analog point to be defined in any length necessary.

#### 5-20.6.1.3 Owner Vendor ID and Owner Serial Number Attributes

These attributes may be implemented in both the point and group objects. If implemented in both, the values shall be the same for the point objects and the group object that binds the point object. If a connection to this object is made using the Connection Manager, then the value for the Owner Vendor Id and Owner Serial Number shall be the Originator Vendor Id and Originator Serial Number contained in the Forward\_Open request. These values shall be cleared when the connection is closed or deleted.

#### 5-20.6.1.4 Temp Mode Attribute

The attribute Temp Mode is a Boolean data type that governs the behavior of the Value attribute of the AIP when returning a temperature. The temperature is returned in either Celcius or Fahrenheit according to the current value of the Temp Mode attribute. AIP values are typically returned as a temperature for thermocouple and RTD channels when linearization is turned on.

## **5-21      Analog Output Group Object**

**Class Code: 21 hex**

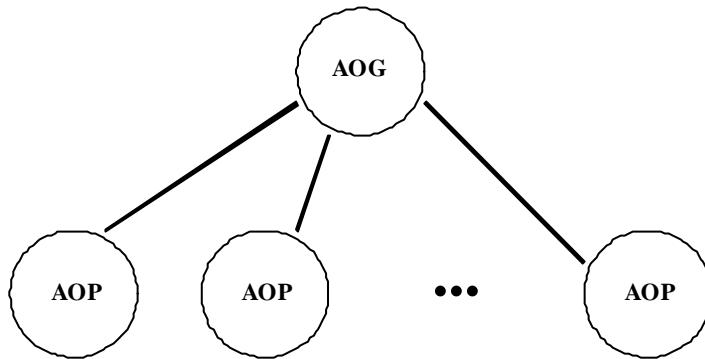
The Analog Output Group Object (AOG) binds a group of analog output points in a module. All points bound to the group share all attributes contained in the group. If an attribute is shared (points have the same attributes and the same attribute values) across more than one Analog Output Point (AOP), then that attribute can be contained in an Analog Output Group. An Analog Output Point can be bound to more than one Analog Output Group.

You must establish the list of analog output points bound to the group at power-up, as the binding list is static and is a Get-only attribute of the group.

When to use the Analog Output Group Object:

- when a single attribute is shared among many input points.
  - to more efficiently access data (services that affect all members of a group can be supported by the AOG).

## **Figure 5-21.1 Analog Output Group Object Application**



## 5-21.1 Class Attributes

**Table 5-21.2 Analog Output Group Object Class Attributes**

Number	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1	Required	Get	Revision	UINT	Revision of this object	The current value assigned to this attribute is two (02)
2 thru 7	These class attributes are optional and are described in Chapter 4 of this specification.					

## 5-21.2 Instance Attributes

**Table 5-21.3 Analog Output Group Object Instance Attributes**

Attr ID	Need in Implem	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1	Optional	Get	Num Attributes	USINT	Number of attributes supported by the group	
2	Optional	Get	Attribute List	Array of USINT	List of attributes supported by the group	
3	Optional	Get	Number of Bound Instances	USINT	Number of bindings in the group	
4	Required if any Analog Output Points are not included in the Analog Output Group	Get	Bound Instances	ARRAY of UINT	List of all points bound to this group	
5	Optional	Get	Status	BOOL	Group is operating without alarms or faults	0 = good; 1 = alarm state
6	Optional	Get	Owner - Vendor ID	UINT	Vendor ID of group's owner	
7	Optional	Get	Owner - Serial Number	UDINT	32-bit serial number of group's owner	
8	Optional	Set	Value Data Type	USINT	Determines the data type of any bound Values	Default = 0 = INT 0 = INT 1 = REAL 2 = USINT 3 = SINT 4 = DINT 5 = LINT 6 = UINT 7 = UDINT 8 = ULINT 9 = LREAL 100=vendor specific
9	Required	Set	Command	BOOL	Changes state of AOP to Idle Mode or Run Mode	0 = idle 1 = run
10	Optional	Set	Fault Action	USINT	Output value to go to on failure or fault	0 = use Fault 1 = hold last state (default) 2 = low limit 3 = high limit 4 – 99 = reserved 100 – 199 = vendor specific 200 – 255 = reserved
11	Required if Fault Action is Implemented	Set	Fault Value	INT or based on attribute 8	User-defined value for use with Fault State attribute	

**Analog Output Group Object, Class Code: 21<sub>Hex</sub>**

Attr ID	Need in Implem	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
12	Optional	Set	Idle Action	USINT	Output value to go to on Idle	0 = use Idle Value <sup>1</sup> 1 = hold last state (default) 2 = low limit 3 = high limit 4 – 99 = reserved 100 – 199 = vendor specific 200 – 255 = reserved
13	Required if Idle Action is implemented	Set	Idle Value	INT or based on attribute 8	User-defined value for use with Idle State attribute	

<sup>1</sup> All values 0, 1, 2, and 3 shall be supported when the Fault Action or Idle Action attributes are implemented.

## 5-21.3 Semantics

### 5-21.3.1 Command, Fault Action, Fault Value, Idle Action, Idle Value, Value Data Type

When any of these attributes are implemented at the Group level, the corresponding attribute of the point objects bound to the group shall assume the value of the Group level object attribute. The attribute shall be implemented at the point level and shall be “Get” only.

The Idle Action and Idle Value behavior shall be the power up default behavior and the behavior when in the available state.

### 5-21.3.2 Value Data Type

When this attribute is implemented at the Group level, the corresponding attribute of the point objects bound to the group shall assume the value of the Group level object attribute. The attribute shall be implemented at the point level and shall be “Get” only.

## 5-21.4 Common Services

The Analog Output Group Object provides the following Common Services:

**Table 5-21.4 Analog Output Group Object Common Services**

Service Code	Need In Implementation		Service Name	Description of Service
	Class	Instance		
0E hex	Conditional <sup>1</sup>	Required	Get_Attribute_Single	Returns the contents of the specified attribute
10 hex	N/A	Required <sup>2</sup>	Set_Attribute_Single	Modifies an attribute value
01 hex	Optional	Optional	Get_Attributes_All	Returns a predefined listing of this object's attributes (See the Get_Attributes_All_Response definition below)
02 hex	N/A	Optional	Set_Attributes_All	Modifies the contents of the attributes of the class or object.

<sup>1</sup> The Get\_Attribute\_Single service is REQUIRED if any attributes are implemented.

<sup>2</sup> The Set\_Attribute\_Single service is required only if Instance Attributes #8 and/or #10 are implemented.

See Appendix A for definitions of these services.

### 5-21.4.1 Get\_Attributes\_All Response

At the **Class level**, the order of the attributes returned in the “Object/service specific reply data” portion of the Get\_Attributes\_All response is as follows:

**Table 5-21.5 Get\_Attributes All Response Data – Class Level**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0								Revision (low byte) Default = 1
1								Revision (high byte) Default = 0
2								Max Instance (low byte) Default = 0
3								Max Instance (high byte) Default = 0
4								Max ID Number of Class Attributes (low byte) Default = 0
5								Max ID Number of Class Attributes (high byte) Default = 0
6								Max ID Number of Instance Attributes (low byte) Default = 3
7								Max ID Number of Instance Attributes (high byte) Default = 0

**Important:** Insert default values for all unsupported attributes.

At the **Instance level**, the order of the attributes returned in the “Object/service specific reply data” portion of the Get\_Attributes\_All response is as follows:

**Table 5-21.6 Get\_Attributes All Response Data – Instance Level**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	0	0	0	Command
1								Number of Attributes Default = 0
2								Attribute List (attribute #1)
n								Attribute List (attribute #m)
n+1								Number of Bound Instances Default = 0
.								Binding : Instance ID #1 (low byte)
.								Binding : Instance ID #1 (high byte)
.								Binding : Instance ID #m (low byte)
.								Binding : Instance ID #m (high byte)
.	0	0	0	0	0	0	0	Status Default = 0
.								Owner Vendor ID (low byte) Default = 0
.								Owner Vendor ID (high byte) Default = 0
.								Owner Serial Number (low byte) Default = 0
.								Owner Serial Number Default = 0
.								Owner Serial Number (high byte) Default = 0
.								Value Data Type Default = 0
.								Fault Action, Default = 1
.								Fault Value (low byte) Default = 0
.								Fault Value (high byte) Default = 0
.								Idle Action, Default = 1

**Analog Output Group Object, Class Code: 21<sub>Hex</sub>**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
.	Idle Value (low byte) Default = 0							
.								
.	Idle Value (high byte) Default = 0							

**Important:** Insert default values for all unsupported attributes.

**Important:** If the instance attribute "Number of Attributes" is not supported, the default value of zero is to be inserted in its place and **no** members of the "Attribute List" attribute will follow.

**Important:** If the instance attribute "Number of Bound Instances" is not supported, the default value of zero is to be inserted into the response byte array and **no** Binding data will follow.

#### 5-21.4.2 Set\_Attributes\_All Request

No settable attributes currently exist at the **Class level** for the Analog Output Group Object.

At the **Instance level**, the order of attributes passed in the "Object/service specific request data" portion of the Set\_Attributes\_All request is as follows:

**Table 5-21.7 Set\_Attributes All Request Service Data – Instance Level**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	0	0	0	Command
1	Value Data Type							
2	0	0	0	0	0	0	0	Fault State
3	Fault Value (low byte) Default = 0							
n	Fault Value (high byte) Default = 0							
N+1	0	0	0	0	0	0	0	Idle State
.	Idle Value (low byte) Default = 0							
.	Idle Value (high byte) Default = 0							

**Important:** The Set\_Attributes\_All service is to be supported only if **all** settable attributes shown above are implemented as settable.

#### 5-21.5 Object-specific Services

The Analog Output Group Object provides no Object-specific services.

#### 5-21.6 Behavior

The primary purpose of the Analog Output Group is to bind Analog Output Points.

An attribute of the AOG will modify the behavior or report additional status information of all AOPs bound to the group.

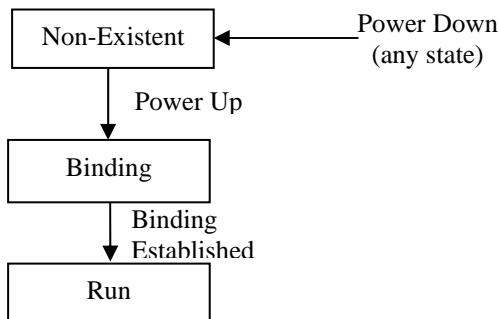
**Analog Output Group Object, Class Code: 21<sub>Hex</sub>**

The State Transition Diagram below provides a graphical description of the events and corresponding state transitions. A subset of the states and events may be supported in an application, but the behavior must be consistent.

The states shown are equivalent to the following:

**Table 5-21.8 Analog Output Group Object State Definitions**

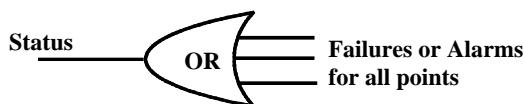
<b>This state</b>	<b>Is equivalent to</b>
Non-Existent	A module without power
	A module that has an unrecoverable fault (e.g. processor watchdog timeout)
	A major reconfiguration of the module (e.g. NVS Update)
Binding	When analog output points are added or bound to the Analog Output Group (executed as part of Power Up cycle)
Run	The point at which Get and Set attribute services can be used to access the attributes of the Group Object

**Figure 5-21.9 State Transition Diagram for Analog Output Group Object****5-21.6.1 Attribute Access Rules**

All attributes are gettable or settable according to their attribute access rules, but only in the Run state.

**5-21.6.1.1 Status Attribute**

The *Status* attribute is simply a logical OR of all possible failure or alarm conditions for all points bound to the group.

**Figure 5-21.10 Analog Output Group Object Status Attribute Function****5-21.6.1.2 Value Data Type Attribute**

The *Value Data Type* attribute determines the data type to be used by the attribute Value (and perhaps other attributes as well). If Value Data Type is not used, then Value defaults to INT. Value (and other attributes) may behave as REAL, USINT, or any other data type based upon the definition of Value Data Type, which ultimately provides the ability for the analog point to be defined in any length necessary.

**5-21.6.1.3 Owner Vendor ID and Owner Serial Number Attributes**

These attributes may be implemented in both the point and group objects. If implemented in both, the values shall be the same for the point objects and the group object that binds the point object. If a connection to this object is made using the Connection Manager Object, then the value for the Owner Vendor Id and Owner Serial Number shall be the Originator Vendor Id and Originator Serial Number contained in the Forward\_Open request. These values shall be cleared when the connection is closed or deleted.

**5-21.6.1.4 Fault Action/Idle Action Values for Analog Output Group and Analog Output Point****Table 5-21.11 Corresponding Attribute Values for Output Group/Analog Output Point Objects**

<b>Value of Analog Output Group Instance Attributes #10 and #12</b>	<b>Meaning</b>	<b>Value of Corresponding Analog Output Point Instance Attributes #9 and #10</b>
0	Use Fault/Idle Value	3
1	Hold Last State (default)	0
2	Use Low Limit	1
3	Use High Limit	2

## 5-22 Analog Group Object

**Class Code: 22 hex**

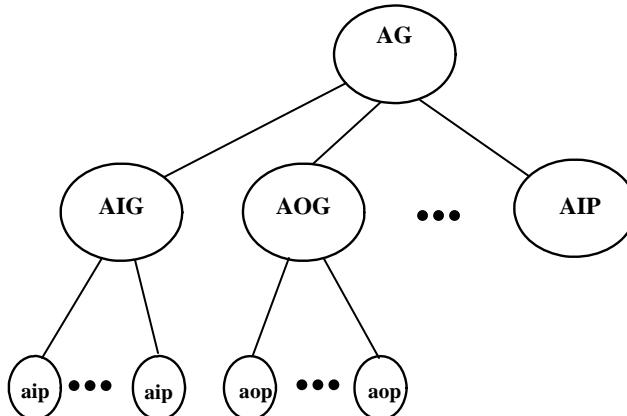
The Analog Group (AG) Object binds other analog objects including AIP, AOP, AIG or AOG. The objects (both inputs and outputs) bound to the Analog Group must support the same attributes and services.

You must establish the list of analog groups and/or points bound to the Analog Group at power-up, as the Binding List is static and a Get-only attribute of the Analog Group.

When to use the Analog Group Object:

- when a single attribute is shared among many analog input and output points and/or groups.
- to more efficiently access data (services that affect all members of a group and/or points can be supported by the AG).

**Figure 5-22.1 Analog Group Object Application**



### 5-22.1 Class Attributes

**Table 5-22.2 Analog Group Object Class Attributes**

Number	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1 thru 7	These class attributes are optional and are described in Chapter 4 of this specification.					

## 5-22.2 Instance Attributes

Table 5-22.3 Analog Group Object Instance Attributes

Attr ID	Need in Implem	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1	Optional	Get	Number of Attributes	USINT	Number of attributes supported by the group	
2	Optional	Get	Attribute List	ARRAY of USINT	List of attributes supported	
3	Optional	Get	Number of Bound Instances	USINT	Number of points in a group	
4	Optional	Get	Binding	ARRAY of STRUCT:	List of all instances in group	
			Class ID	UINT		
			Instance ID	UINT		
5	Optional	Get	Status	BOOL	Group is operating without alarms or faults	0=good; 1=alarm state
6	Optional	Get	Owner - Vendor ID	UINT	Vendor ID of group's owner	
7	Optional	Get	Owner - Serial Number	UDINT	32-bit serial number of group's owner	
8	Optional	Set	Value Data Type	BYTE	Determines the data type of all bound object's Value which propagate through another binding if groups are bound to AG	0 = INT 1 = REAL 2 = USINT 3 = SINT 4 = DINT 5 = LINT 6 = UINT 7 = UDINT 8 = ULINT 9 = LREAL 100 = vendor specific

## 5-22.3 Common Services

The Analog Group Object provides the following Common Services:

Table 5-22.4 Analog Group Object Common Services

Service Code	Need In Implementation		Service Name	Description of Service
	Class	Instance		
0E hex	Conditional <sup>1</sup>	Required	Get_Attribute_Single	Returns the contents of the specified attribute
10 hex	N/A	Required <sup>2</sup>	Set_Attribute_Single	Modifies an attribute value
01 hex	Optional	Optional	Get_Attributes_All	Returns a predefined listing of this object's attributes (See the Get_Attributes_All_Response definition below)

<sup>1</sup> The Get\_Attribute\_Single service is REQUIRED if any attributes are implemented.

<sup>2</sup> The Set\_Attribute\_Single service is required only if Instance Attributes #8 and/or #10 are implemented.

See Appendix A for definitions of these services.

### 5-22.3.1 Get\_Attributes\_All Response

At the **Class level**, the order of the attributes returned in the “Object/service specific reply data” portion of the Get\_Attributes\_All response is as follows:

**Table 5-22.5 Get\_Attributes All Response Data – Class Level**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0								Revision (low byte) Default = 1
1								Revision (high byte) Default = 0
2								Max Instance (low byte) Default = 0
3								Max Instance (high byte) Default = 0
4								Max ID Number of Class Attributes (low byte) Default = 0
5								Max ID Number of Class Attributes (high byte) Default = 0
6								Max ID Number of Instance Attributes (low byte) Default = 0
7								Max ID Number of Instance Attributes (high byte) Default = 0

**Important:** Insert default values for all unsupported attributes.

At the **Instance level**, the order of the attributes returned in the “Object/service specific reply data” portion of the Get\_Attributes\_All response is as follows:

**Table 5-22.6 Get\_Attributes\_All Response Data – Instance Level**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	0	0	0	Command
1								Number of Attributes Default = 0
2								Attribute List (attribute #1)
n								Attribute List (attribute #m)
n+1								Number of Bound Instances Default = 0
.								Binding: Class ID #1 (low byte)
.								Binding: Class ID #1 (high byte)
.								Binding: Instance ID #1 (low byte)
.								Binding: Instance ID #1 (high byte)
.								Binding: Class ID #m (low byte)
.								Binding: Class ID #m (high byte)
.								Binding: Instance ID #m (low byte)
.								Binding: Instance ID #m (high byte)
.	0	0	0	0	0	0	0	Status Default = 0
.								Owner Vendor ID (low byte) Default = 0
.								Owner Vendor ID (high byte) Default = 0
.								Owner Serial Number (low byte) Default = 0
.								Owner Serial Number Default = 0
.								Owner Serial Number Default = 0
.								Owner Serial Number (high byte) Default = 0
.								Value Data Type Default = 0

**Important:** Insert default values for all unsupported attributes.

**Important:** If the instance attribute "Number of Attributes" is not supported, the default value of zero is to be inserted in its place and **no** members of the "Attribute List" attribute will follow.

**Important:** If the instance attribute "Number of Bound Instances" is not supported, the default value of zero is to be inserted into the response byte array and **no** Binding data will follow.

## 5-22.4 Object-specific Services

The Analog Group Object provides no Object-specific services.

## 5-22.5 Behavior

The primary purpose of the Analog Group Object is to bind analog groups and/or points.

An attribute of the AG modifies the behavior or reports additional status information of all objects bound to the group.

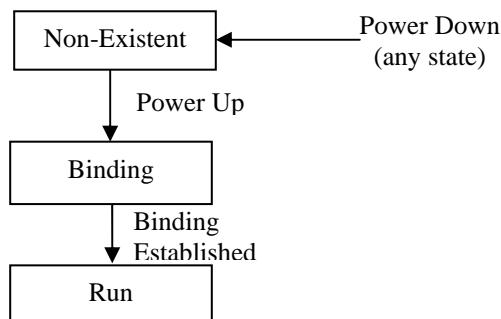
The State Transition Diagram below provides a graphical description of the events and corresponding state transitions. A subset of the states and events may be supported in an application, but the behavior must be consistent.

Except for highly configurable modules (which could have explicit Create and Delete events), the states shown in the STD are equivalent to the following:

**Table 5-22.7 Analog Group Object State Definitions**

This state	Is equivalent to
Non-Existent	a module without power a module that has an unrecoverable fault (e.g. processor watchdog timeout) a major reconfiguration of the module (e.g. NVS Update)
Binding	when the analog input and/or output groups and/or points are added or bound to the Analog Group (executed as part of Power Up cycle)
Run	the point at which Get and Set attribute services can be used to access the attributes of the Analog Group

**Table 5-22.8 Analog Group Object State Transition Diagram**



### 5-22.5.1 Attribute Access Rules

All attributes are gettable or settable according to their attribute access rules, but only in the Run state.

#### 5-22.5.1.1 Status Attribute

The *Status* attribute is simply a logical OR of all possible failure or alarm conditions for all points bound to the group.

**Table 5-22.9 Analog Group Object Status Attribute Function**



#### 5-22.5.1.2 Value Data Type Attribute

The *Value Data Type* attribute determines the data type to be used by all of the bound AIP's Value attribute (and perhaps other attributes as well). If Value Data Type is not used, then Value defaults to INT. Value (and other attributes) may behave as REAL, USINT, or any other data type based upon the definition of Value Data Type, which ultimately provides the ability for the analog point to be defined in any length necessary.

#### 5-22.5.1.3 Owner Vendor ID and Owner Serial Number Attributes

These attributes may be implemented in both the point and group objects. If implemented in both, the values shall be the same for the point objects and the group object that binds the point object. If a connection to this object is made using the Connection Manager, then the value for the Owner Vendor Id and Owner Serial Number shall be the Originator Vendor Id and Originator Serial Number contained in the Forward\_Open request. These values shall be cleared when the connection is closed or deleted.

## 5-23 Position Sensor Object

**Class Code: 23 hex**

The Position Sensor Object models an absolute position sensor in a product. Behaviors in the object extend the basic position sensor capability to include zero offset, and position boundary checking (CAM switch).

The Position Sensor Object interface is to real position sensor hardware such as an absolute digital encoder, an analog resolver or other absolute position-input device.

### 5-23.1 Revision History

Since the initial release of this object class definition changes have been made that require a revision update of this object class. The table below represents the revision history.

**Table 5-23.1 Revision History**

Revision	Reason for Change
01	Initial release of this object class
02	Add new attributes

### 5-23.2 Class Attributes

**Table 5-23.2 Position Sensor Object Class Attributes**

Attr ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1	Required	Get	Revision	UINT	Revision of this object	The current value assigned to this attribute is 02.
2 thru 7	These class attributes are optional and are described in Chapter 4 of this specification.					

### 5-23.3 Instance Attributes

**Table 5-23.3 Position Sensor Object Instance Attributes**

Attr ID	Need in Implem	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
1	Optional	Get	NV	Number of Attributes	USINT	Number of attributes supported in this product	
2	Optional	Get	NV	Attribute List	Array of USINT	List of attributes supported in this product.	
3	Conditional <sup>1</sup>	Get	V	Position Value Unsigned	UDINT	Current position.	Physical position See Semantics Section
4	Optional	Get	V	CAM	BOOL	Virtual CAM switch value	0 = Off 1 = On
5	Optional	Set	V	Value Bit Resolution	USINT	Position sensor resolution.	See Semantics.
6	Optional	Set	V	Zero Offset	UDINT	Value attribute zero offset	See Semantics.
7	Optional	Set	V	CAM Low Limit	UDINT	Virtual CAM switch low limit.	The default is 0

Position Sensor Object, Class Code: 23<sub>Hex</sub>

Attr ID	Need in Implem	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
8	Optional	Set	V	CAM High Limit	UDINT	Virtual CAM switch high limit.	The default is 0
9	Optional	Set	V	Auto Zero	BOOL	Auto zero control.	Rising edge sets Zero Offset to Position Value
10	Conditional <sup>1</sup>	Get	V	Position Value Signed	DINT	Current position	The content is based upon 14, 15, 16, and 17.
11	Optional	Get	NV	Position Sensor Type	UINT	Specifies the device type	See Semantics section
12	Required	Set	NV	Direction Counting Toggle	BOOL	Defines the direction of increasing 'Position Value'	Default = 0. See semantics section
13	Optional	Set	NV	Commissioning Diagnostic Control	BOOL	Check encoder at encoder stand still	0 = OFF 1 = ON (Default)
14	Optional	Set	NV	Scaling Function Control	BOOL	<i>Physical resolution span</i> (attribute 42) is converted to a numerical value	0 = OFF 1 = ON (Default)
15	Optional	Set	NV	Position Format	ENGUNIT	Format of the position value of other attributes.	Supported units: counts(default) millimeter micron nanometer inch thousandths inch ten thousandths inch 0x0800-0xFFFF = vendor specific
16	Optional	Set	NV	Measuring Units per Span	UDINT	Number of distinguishable steps per one complete span. Less than or equal to <i>Physical Resolution Span</i> (attribute 42).	For rotary devices a span equals one revolution.
17	Optional	Set	NV	Total Measuring Range in Measuring Units	UDINT	Steps over the total measuring range. Only used for rotary encoders.	
18	Optional	Set	NV	Position Measuring Increment	UDINT	Specifies the smallest incremental change of the <i>Position Value Signed</i> or <i>Position Value Unsigned</i> attributes	Units depend on <i>Position Format</i> attribute. Default = 1.
19	Optional	Set	NV	Preset Value	DINT	Output position value is set to Preset Value.	See Semantics section
20	Optional	Set	NV	COS/delta	UDINT	Value for position change in COS mode.	See Semantics section
21	Optional	Get	V	Position State Register	BYTE	The state of software limit switch.	Bit 0: 1 = Out of Range Bit 1: 1 = Range overflow Bit 2: 1 = Range underflow
22	Optional	Set	NV	Position Low Limit	DINT	Low Limit Position	See Semantics section

Position Sensor Object, Class Code: 23<sub>Hex</sub>

Attr ID	Need in Implem	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
23	Optional	Set	NV	Position High Limit	DINT	High Limit Position When attribute 10 is greater than this value, the <i>Position State Register</i> , bit 1 shall be set to one (1).	See Semantics section
24	Optional	Get	V	Velocity Value	DINT	Current speed where the format of this value is defined in attributes 25 & 26.	Default is attribute 15 per second  The meaning of the sign is affected by the value of attribute 12. See Semantics section
25	Conditional	Set	NV	Velocity Format	ENGUINT	Format of the velocity attributes.	0x1f04 = counts (Steps) per second (default)  See Semantics section
26	Conditional	Set	NV	Velocity Resolution	UDINT	Specifies the smallest incremental change of the <i>Velocity Value</i> attribute 24.	Units depend on Attr 25 Default = 1.  See Semantics section
27	Conditional	Set	NV	Minimum Velocity Setpoint	DINT	Value for minimum velocity trigger threshold. Affects Min. Velocity Flag in status <i>Warning</i> Attribute 47	Default = 0x80000000.  See Semantics section
28	Conditional	Set	NV	Maximum Velocity Setpoint	DINT	Value for maximum velocity trigger threshold. Affects Max. Velocity Flag in status <i>Warning</i> Attribute 47	See Semantics section Default = 0xFFFFFFFF.  See Semantics section
29	Optional	Get	V	Acceleration Value	DINT	Current Acceleration where the format of this value is defined in attributes 30 & 31.	Default is <i>Velocity Value</i> attribute 24 per second <sup>2</sup> Positive value is acceleration, negative is deceleration.  See Semantics section
30	Conditional	Set	NV	Acceleration Format	ENGUINT	Format of the Acceleration attributes.	0x1500 = m/s <sup>2</sup> (default)  See Semantics section
31	Conditional	Set	NV	Acceleration Resolution	UDINT	Specifies the smallest incremental change of the <i>Acceleration Value</i> , attribute 29	Default = 1.  See Semantics section
32	Conditional	Set	NV	Minimum Acceleration Setpoint	DINT	Value for minimum acceleration trigger threshold.	Default = 0x8000 0000.  See Semantics section
33	Conditional	Set	NV	Maximum Acceleration Setpoint	DINT	Value for maximum acceleration trigger threshold.	Default = 0xFFFF FFFF .  See Semantics section
34	Optional	Get	NV	Number of CAM Channels	USINT	Contains the number of independent cams	See Semantics section
35	Conditional	Get	V	CAM Channel State register	ARRAY of BOOL	Contains state of independent cam channels	Bit 0 = CAM_ 1 State ... Bit x = CAM x+1

Position Sensor Object, Class Code: 23<sub>Hex</sub>

Attr ID	Need in Implem	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
36	Conditional	Set	NV	CAM Channel Polarity Register	ARRAY of BOOL	Determines the polarity for each <i>Cam Channel State Register</i>	Bit 0 = CAM_1 Polarity ... Bit x = CAM x+1
37	Conditional	Set	NV	CAM Channel Enable Register	ARRAY of BOOL	Enables each independent cam channel where size of array is equal to <i>Number of CAM Channels</i> attribute.	Bit 0 = CAM_1 Enable ... Bit x = CAM x+1
38	Conditional	Set	NV	CAM Low Limit	ARRAY of DINT	Switch point for the lower limit where size of array is equal to <i>Number of CAM channels</i> attribute. When <i>Position Value Signed</i> is less than <i>CAM Low Limit</i> value, then <i>CAM Channel State Register</i> = 1.	
39	Conditional	Set	NV	CAM High Limit	ARRAY of DINT	Switch point for the higher limit where size of array is equal to “Number of CAM channel” <i>Number of CAM channels</i> attribute. <i>Position Value Signed</i> is greater than <i>CAM High Limit</i> value, then <i>CAM Channel State Register</i> = 1.	
40	Conditional	Set	NV	CAM Hysteresis	ARRAY of UINT	This value will be added to the <i>CAM High Limit</i> and subtracted from the <i>CAM Low Limit</i> when calculating the cam state.	See Semantics section
41	Optional	Get	V	Operating Status	BYTE	Encoder diagnostic operating status	See Semantics Section
42	Optional	Get	NV	Physical Resolution Span	UDINT	Number of distinguishable steps per one complete span.	For rotary devices, a span equals one revolution.
43	Optional	Get	NV	Number of Spans	UINT	This is equal to the number of turns when a rotary type device is used.	Default = 1.
44	Optional	Get	V	Alarms	WORD	Indicates a malfunction has occurred that could lead into an incorrect position value or require user intervention	See Semantics Section
45	Conditional	Get	NV	Supported Alarms	WORD	Information about supported <i>Alarms</i>	See Semantics Section
46	Conditional	Get	V	Alarm Fag	BOOL	Indicates that an alarm error occurred.	See Semantics Section 0 = OK 1 = Alarm error

Position Sensor Object, Class Code: 23<sub>Hex</sub>

Attr ID	Need in Implem	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
47	Optional	Get	V	Warnings	WORD	Internal parameters exceeded	See semantics section.
48	Conditional	Get	NV	Supported Warnings	WORD	Information about supported <i>Warnings</i>	See Semantics Section
49	Conditional	Get	V	Warning Flag	BOOL	Indicates that a warning error occurred	See Semantics Section 0 = OK 1 = Warning flag
50	Optional	Get	NV	Operating Time	UDINT	Stores operating time for the encoder in tenths of an hour	
51	Conditional	Get	NV	Offset Value	DINT	The Offset value is calculated by the preset function. Shift position value with the calculated value	See Semantics Section

1 One and only one of the attributes *Position Value Signed* or *Position Value Unsigned* shall be implemented depending on the application requirements.

## 5-23.4 Semantics

### 5-23.4.1 Position Value Unsigned - Attribute 3

This attribute represents the absolute position detected by the position sensor conditioned by the *Value Bit Resolution* and *Zero Offset* attributes. Refer to the following descriptions of the *Value Bit Resolution* and *Zero Offset* attributes for details.

### 5-23.4.2 Position Value Signed - Attribute 10

This attribute represents the absolute position detected by the position sensor. It is not conditioned by the *Value Bit Resolution* and *Zero Offset* attributes.

### 5-23.4.3 Value Bit Resolution – Attribute 5

This attribute specifies the number of significant bits used for the *Position Value Unsigned* (attribute 3). The raw value is shifted left or right to supply the indicated number of significant bits.

**Table 5-23.4 Bit Resolution Attribute Value**

Resolution	Value =
> Physical Resolution	(RawValue << (PhysicalResolution - Bit Resolution)) + ZeroOffset
< Physical Resolution	(RawValue >> (Bit Resolution - PhysicalResolution)) + ZeroOffset
= Physical Resolution	RawValue + ZeroOffset

**Table 5-23.5 Example Bit Resolution Values**

<b>Raw Value</b>	<b>Resolution</b>	<b>Adjusted Val.</b>	<b>Notes</b>
10 bit (0 to 3FF <sub>hex</sub> )	8	0 to FF <sub>hex</sub>	Bit Resolution < Physical Resolution, surplus bits are discarded
6 bit (0 to 3F <sub>hex</sub> )	8	0 to FF <sub>hex</sub>	Bit Resolution > Physical Resolution, missing bits are zero. Adjusted value will actually be 0 to FC <sub>hex</sub> in multiples of 4
10 bit (0 to 3FF <sub>hex</sub> )	10	0 to 3FF <sub>hex</sub>	Bit Resolution = Physical Resolution, no conversion.

**5-23.4.4 Zero Offset – Attribute 6**

The Zero Offset Attribute adjusts the zero point of Value. Zero Offset is added to Value to adjust the zero point. The Zero Offset Attribute is applied *after* the *Value Bit Resolution* Attribute.

If the result of the addition exceeds the maximum specified by the Resolution attribute the overflow bits are discarded.

**Table 5-23.6 Example Zero Offset Values**

<b>Adjusted Val.</b>	<b>Zero Offset</b>	<b>Resolution</b>	<b>Value</b>
0	10	8	10
250	0	8	250
250	20	8	15 <sup>1</sup>
250	20	10	270
250	255	8	249 <sup>2</sup>

1 Value overflowed

2 Value underflowed

**5-23.4.4.1 CAM, CAM Low Limit, CAM High Limit – Attributes 4, 7 and 8**

The *CAM* attribute is a virtual CAM switch. The state of the *CAM* attribute is determined by the *CAM Low Limit*, *CAM High Limit* and *Position Value Unsigned* attributes. The *Position Value Unsigned* attribute is used after the *Value Bit Resolution* and *Zero Offset* attributes have been applied.

**Table 5-23.7 CAM, CAM Low Limit & CAM High Limit Operation**

<b>CAM Low Limit</b>	<b>CAM is On (true, 1) if ...</b>	<b>CAM is Off (false, 0) if ...</b>
> CAM High Limit	Value > CAM Low <i>or</i> Value < CAM High	Value < CAM Low <i>and</i> Value > CAM High
< CAM High Limit	Value > CAM Low <i>and</i> Value < CAM High	Value < CAM Low <i>or</i> Value > CAM High
= CAM High Limit	Never	Always

**5-23.4.5 Auto Zero – Attribute 9**

This attribute controls the auto-zero feature of the resolver. A rising edge (transition from 0 to 1) on this attribute adjusts the *Zero Offset* attribute to a value that results in the *Position Value Unsigned* attribute being zero.

If the *Zero Offset* attribute is implemented as non-volatile, the AutoZero command must store the new *Zero Offset* value.

**5-23.4.6 Position Sensor Type – Attribute 11****Table 5-23.8 Attribute 11, Position Sensor Type Values**

Value	Definition
00	Single Turn resolver (value if attribute is not supported)
01	Single-Turn absolute rotary encoder
02	Multi-Turn absolute rotary encoder
03	Single-Turn absolute rotary encoder with electronic turn count
04	Incremental rotary encoder
05	Incremental rotary encoder with electronic counting
06	Incremental linear encoder
07	Incremental linear encoder with electronic counting
08	Absolute linear encoder
09	Absolute linear encoder with cyclic coding
10	Multi-Sensor encoder interface
11	Multi-Turn absolute rotary encoder with electronic turn count
12	Virtual Axis Sensor
13... 65535	Reserved by CIP

**5-23.4.7 Direction Counting Toggle – Attribute 12**

For linear devices, the *Position Value* attribute increases when the value of the *Direction Counting Toggle* attribute is FORWARD/CW (0). If the value of the *Direction Counting Toggle* attribute is REVERSE/CCW (1), the *Position Value* attribute decreases. The *Direction Counting Toggle* defines the increasing *Position Value* 1) for rotary devices as clockwise shaft rotation or counter clockwise as viewed facing the mounting face (CW = 0, CCW = 1), 2) for linear devices as Forward, moving away from the electrical connection, or Reverse, moving toward the electrical connection, as viewed from the mounting face. Changing this value shall change the sign of velocity and position relative to physical movement.

For rotary encoders the code sequence defines whether increasing or decreasing position values are output when the encoder shaft rotates clockwise (CW) or counterclockwise (CCW) as seen on the shaft. Example: By turning the shaft clockwise the *Position Value* attribute will increase when *Direction Counting Toggle* is defined as clockwise (0=CW).

**5-23.4.8 Commissioning Diagnostic Control – Attribute 13**

When the *Commissioning Diagnostic Control* attribute is set to ON (1) it is possible to check the encoder components responsible for position detection at encoder stand still. This enables an extensive check of the correctness of the position values.

If errors are detected, they will be indicated by the respective bits in the *Alarm* attribute. If this attribute is set to OFF (0), no diagnostics will be executed.

#### **5-23.4.9 Scaling Function Control – Attribute 14**

When the *Scaling Function Control* attribute is set to ON (1), the *Position Value* attribute is converted from the physical resolution of the device to position units.

If this attribute is implemented and turned OFF, all functions within position sensor shall continue to use the scaled value, except the *Position Value* (attribute 10) reported shall be the raw, unscaled value. This attribute is provided solely for calibration and troubleshooting purposes and does not affect scaling when *Position Format* (attribute 15) and *Measuring Units per Span* (attribute 16) are implemented.

The *Measuring Units per Revolution* and *Total Measuring Range in Measuring Units* attributes are the scaling parameters.

For rotary devices:

If *Scaling Function Control* == OFF

Then *Position Value* = physical resolution of device in counts

If *Scaling Function Control* == ON

The *Position Value* = physical resolution of device in counts \* (*Measuring Units per Span* (attribute 16) / *Physical Resolution Span* (attribute 42))

If *Measuring Units per Span* and *Total Measuring Range in Measuring Units* are not supported, the scaling of the *Position Value* is not provided and the *Position Value* is always equal to physical resolution of the device in counts.

If *Scaling Function Control* is not supported, the *Position Value* shall always be scaled based upon the values within *Position Format* and *Measuring Units per Span* attributes. When the *Scaling Function Control* attribute is supported and is enabled (ON=1), *Position Value* is calculated according to the formula above. When disabled, physical resolution shall be contained in *Position Value*.

#### **5-23.4.10 Position Format – Attribute 15**

This attribute identifies the engineering units for the *Position Value* attribute (attribute 3 or 10). *Position Format* is a component of all other attributes containing a distance data type, like *Velocity Value* (attribute 24) and *Acceleration Value* (attribute 29). Engineering units like counts, mm or inches are allowed.

The *Position Format* (attribute 3 or 10) identifies the engineering units for one increment of the *Position Value*. The *Position Measuring Increment* (attribute 18) indicates the smallest granularity that the *Position Value* (attribute 3 or 10) shall change.

**5-23.4.11 Measuring Units per Span – Attribute 16**

The *Measuring Units per Span* sets the number of distinguishable (desired) steps per unit of travel. Rotary units would contain the counts per one complete span.

**5-23.4.12 Total Measuring Range in Measuring Units – Attribute 17**

The parameter *Total Measuring Range in Measuring Units* sets the number of distinguishable steps over the total measuring range. This value must be less than maximum physical resolution of the device. Maximum physical resolution should be listed on the type plate.

This parameter is used for rotary and linear devices..

**5-23.4.13 Position measuring increment – Attribute 18**

This attribute may be supported when *Position Format* (attribute 15) can be set to a value other than count (0x1001). The *Position Measuring Step* attribute defines the measuring step settings for the position for linear encoders. Basic position measuring step in 0.001 µm or 0.1nanoinch is affected by *Position Format* (attribute 15).

**5-23.4.14 Preset Value – Attribute 19**

This attribute supports adapting a desired position value to an actual position value.

At the instant a “Set Attribute” to attribute #19 is performed, the following occurs:

Attribute #19 (*Preset Value*) is set to the service data of the request.

*Offset Value* (Attribute #51) is set to the value resulting from [*Preset Value*] - [*Position Value*]

During operation, the formula is immaterial and the behavior of the device is:

$$\text{Position Value} = (\text{internal position value}) + \text{Offset Value}$$

The value contained in the *Preset Value* attribute is meaningless (with respect to the indicated position) after the Set Attribute has taken effect. The *Offset Value* remains a constant and the now reported *Position Value* is the adjusted position value using the offset applied to some (internal) raw position value.

The non-settable attribute (from explicit messaging) is actually set (e.g. the *Offset Value* attribute must change) by an internal mechanism.

$$\text{Preset Value} \text{ (attribute 19)} = \text{Position Value} \text{ (attribute 10)} + \text{Offset Value} \text{ (attribute 51)}.$$

**5-23.4.15 COS Delta – Attribute 20**

A COS I/O message will be generated when the *Positon Value* (attribute 3 or 10) changes by this value. Setting this value to 0 disables COS delta limit for *Position Value* (attribute 3 or 10) changes and all position changes shall generate a COS message.

#### 5-23.4.16 Position State Register, Position limits – Attributes 21 to 23

The *Position Low Limit* and *Position High Limit* attributes configure the actual work area. The *Position State Register* contains the actual area status of the encoder position. If the position is out of range, a bit will be set in the *Position State Register* attribute. If the position is lower than the position value set in *Position Low Limit*, then bit 2 flags the underflow. If the position is higher than the position value set in *Position High Limit*, then bit 1 flags the overflow. The *Position Limits* define a configurable work area within the measuring range. This function allows a replacement of external proximity switches.

**Table 5-23.9 Position State Register Structure**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	Reserved	Reserved	Reserved	Reserved	Range underflow	Range overflow	Out of range

#### 5-23.4.17 Velocity Format– Attribute 25

This *Velocity Format* attribute identifies the engineering units for the *Velocity Value* attribute (attribute 24). Default is counts (steps) per second. Any reasonable format from Appendix D-2.30 may be used depending upon application requirements. *Velocity Format* is a component of all other attributes containing velocity for calculation like acceleration. If this attribute is changed, both *Minimum Velocity* and *Maximum Velocity* Setpoints shall be reset to their default values. This attribute is required when *Velocity Value* is implemented. Otherwise, this attribute is not implemented.

#### 5-23.4.18 Velocity Resolution – Attribute 26

The parameter *Velocity Resolution* defines the resolution of the *Velocity Value* attribute. Basic velocity resolution is defined in steps of 0.01 mm /s or 0.001 inch/s, affected by the chosen *Position Format* (attribute 15). This attribute is required when *Velocity Value* is implemented. Otherwise, this attribute is not implemented.

#### 5-23.4.19 Minimum Velocity / Maximum Velocity - Attribute 27 thru 28

The actual velocity speed limit values with minimum and maximum can be configured in attributes 27 and 28. Corresponding flags in *Status Warnings* (attribute 68) are affected. These attributes are optional when *Velocity Value* is implemented.

#### 5-23.4.20 Acceleration Value - Attribute 29

Acceleration is defined as the change of velocity (speed) per time unit. The time unit used should be seconds. This means the *Acceleration Value* is given in “*Velocity Format*” per second. The current *Acceleration Value* is derived from the calculated *Velocity Value*. The value is signed with the following meaning.

**Table 5-23.10 Acceleration Value Attribute**

Value (sign)	Explanation
0	Velocity is constant
+	Velocity is increasing
-	Velocity is decreasing

**5-23.4.21 Acceleration Format – Attribute 30**

This attribute identifies the engineering units for the *Acceleration Value* (attribute 29). Default is meters/second. Any reasonable format from Appendix D-2.8 may be used depending upon application requirements. If this attribute is changed, both *Minimum Acceleration Setpoint* and *Maximum Acceleration Setpoint* shall be reset to their default values. This attribute is required when *Acceleration Value* is implemented. Otherwise, this attribute is not implemented.

**5-23.4.22 Acceleration Resolution – Attribute 31**

The parameter *Acceleration Resolution* defines the resolution of the *Acceleration Value* attribute. Basic *Acceleration Resolution* is defined in steps of 1 mm/s<sup>2</sup> or 0,1 inch/s<sup>2</sup>, affected by the chosen *Position Format* [attribute 15]. This attribute is required when *Acceleration Value* is implemented. Otherwise, this attribute is not implemented.

**5-23.4.23 Minimum Acceleration / Maximum Acceleration - Attribute 32 thru 33**

The actual acceleration limit values with minimum and maximum can be configured in the attributes 32 and 33. Corresponding flags in status *Warnings* (attribute 47) are affected. These attributes are optional when *Acceleration Value* is implemented. Otherwise, these attributes are not implemented.

**5-23.4.24 Number of CAM Channels – Attribute 34**

This attribute defines the number of cam channels that are supported by an encoder device. Each cam has parameters for the minimum switch point, the maximum switch point and setting a hysteresis to the switch points. If this attribute is implemented then attributes 35-40 shall be implemented. If this attribute is NOT implemented then attributes 35-40 shall NOT be implemented.

Figure 5-23.11 Possible Usage of Cam's and Switch Points

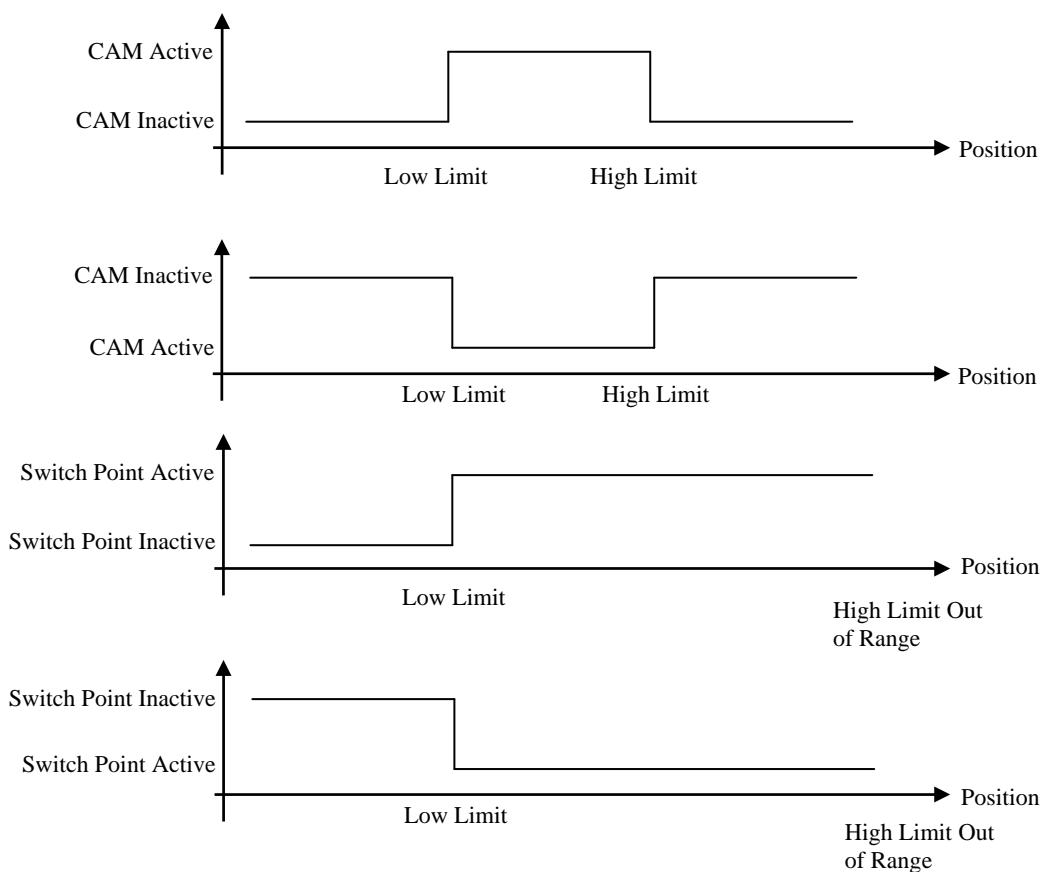
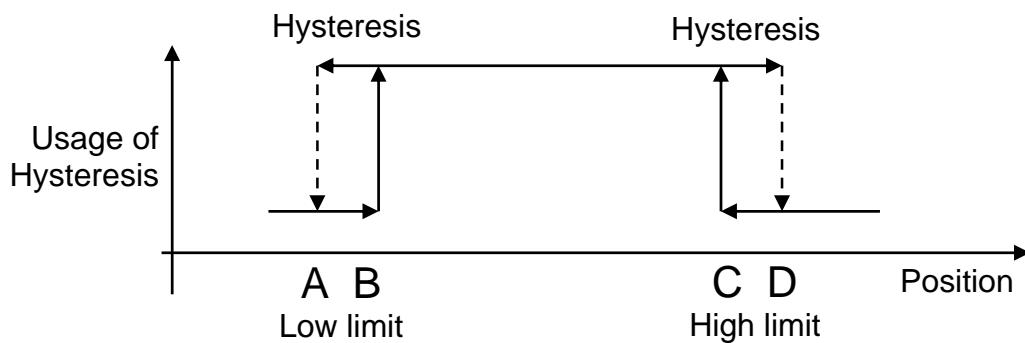
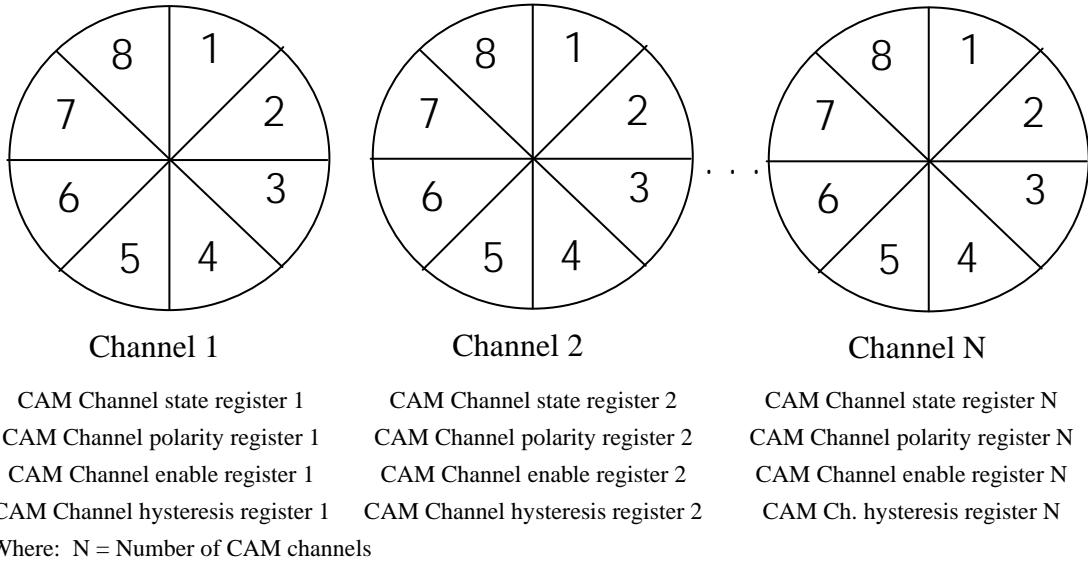


Figure 5-23.12 Possible Usage of Hysteresis



A Low Limit trip occurs at the lower "A" value (dashed left down arrow) and is rest at "B" threshold (the higher solid left up arrow). A High Limit trip occurs at the higher "D" value (dashed right down arrow) and is reset at the lower "C" threshold (solid right up arrow).

Figure 5-23.13 Principle of CAM Channels



#### 5-23.4.25 CAM Channel State Register – Attribute 35

This attribute defines the status bit of the cam. The status bit set to 1 defines “cam active”. The status bit set to 0 defines “cam inactive”. If the *CAM Polarity Register* attribute of a cam is set to one, the actual cam state will be inverted. The *CAM Channel State Register* array size is determined by the value of the *Number of CAM Channels* attribute 34.

Table 5-23.14 Cam Channel State register for CAM Channel N

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
CAM_8	CAM_7	CAM_6	CAM_5	CAM_4	CAM_3	CAM_2	CAM_1
State							

N = Number of CAM channel, with respect to Attribute 34

#### 5-23.4.26 CAM Channel Polarity Register – Attribute 36

This attribute contains the actual polarity settings for *Cam Channel State Register*. If the *CAM Polarity Register* bit is set to 1, the cam state of an active cam will signal by setting the related cam state bit to zero. In the other case, the cam state of the related cam will not be inverted.

Table 5-23.15 CAM Polarity Register of CAM Channel N

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
CAM_8	CAM_7	CAM_6	CAM_5	CAM_4	CAM_3	CAM_2	CAM_1
Polarity							

N = Number of CAM channel, with respect to Attribute 34

**5-23.4.27 CAM Channel Enable Register – Attribute 37**

The *Cam Channel Enable Register* contains the calculation state for the respective cams. If the *Cam Channel Enable Register* bit is set to 1, the cam state will be calculated by the device. In the other case the cam state of the related cam will be set permanently to 0.

**Table 5-23.16 CAM Enable Register of CAM Channel N**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
CAM_8	CAM_7	CAM_6	CAM_5	CAM_4	CAM_3	CAM_2	CAM_1
Enable							

N=Number of CAM channel, with respect to Attribute 34

**5-23.4.28 CAM Low Limit – Attribute 38**

Each cam channel contains the switch point for the lower limit setting for a maximum of 8 cams for one cam channel. The *CAM Low Limit* array size is determined by the value of the *Number of CAM Channels* attribute 34. The default value for this attribute is zero.

**5-23.4.29 CAM High Limit – Attribute 39**

Each cam channel contains the switch point for the higher limit setting for a maximum of 8 cams for one cam channel. The *CAM High Limit* array size is determined by the value of the *Number of CAM Channels* attribute 34. The default value for this attribute is zero.

**5-23.4.30 CAM Hysteresis – Attribute 40**

The *CAM Hysteresis* value is added to the *CAM High Limit* and subtracted from the *CAM Low Limit* when calculating the CAM state. The *CAM Hysteresis* array size is determined by the value of the *Number of CAM Channels* attribute 34.

**5-23.4.31 Operating Status – Attribute 41**

This attribute contains the operating status of the encoder.

**Table 5-23.17 Operating Status Attribute Bit Definitions**

Bit	Description	FALSE (0)	TRUE (1)
0	Direction	Increasing	Decreasing
1	Scaling	Off	On
2..4	Reserved by CIP		
5..7	Vendor specific		

**5-23.4.32 Physical Resolution – Attribute 42**

This is the physical resolution of the position sensor. For rotary devices, the number of steps per span can be read out. For linear devices, the units are steps per linear unit (nanometer or 0.1 nanoinch) the measuring step is given for linear encoder.

**5-23.4.33 Number of Spans – Attribute 43**

For a multi-Turn device the *Number of Spans* and the *Physical Resolution* (attribute 42) gives the physical measuring range to the formula below.

$$\text{Physical Measuring range} = \text{Physical Resolution} * \text{Number of Spans}$$

**5-23.4.34 Alarms – Attribute 44**

An alarm is set if a malfunction bit is set to true (high). The alarm remains active until the alarm is cleared and the device is able to provide an accurate position value.

**Table 5-23.18 Alarms Attribute Bit Definitions**

Bit	Description	FALSE (0)	TRUE (1)
0	Position error	NO	YES
1	Diagnostic error	NO	YES
2...11	Reserved by CIP		
12...15	Vendor specific		

**5-23.4.35 Supported Alarms – Attribute 45**

This attribute contains information on supported alarms by the position sensor device. This attribute is required when the *Alarms* attribute is implemented.

**Table 5-23.19 Supported Alarms Attribute Bit Definitions**

Bit	Description	FALSE (0)	TRUE (1)
0	Position error	Not supported	Supported
1	Diagnostic error	Not supported	Supported
2...11	Reserved by CIP		
12...15	Vendor specific		

**5-23.4.36 Alarm Flag – Attribute 46**

Indicates that an alarm error has occurred. This attribute is the logical OR of all the alarm bits in the *Alarms* attribute (attribute 44). This attribute is required when the *Alarms* attribute is implemented.

**5-23.4.37 Warnings – Attribute 47**

The *Warnings* attribute indicates that tolerance for certain internal parameters of the device have been exceeded. In contrast to alarms, warnings do not imply incorrect position values. All warnings are cleared if the tolerances are again within normal parameters. For the operating time limit warning (bit 3) the warning is only set again after a power-on sequence. The *Warning Flag* attribute indicates if any of the defined warnings are active.

**Table 5-23.20 Warnings Attribute Bit Definitions**

<b>Bit</b>	<b>Description</b>	<b>FALSE (0)</b>	<b>TRUE (1)</b>
0	Frequency Exceeded	NO	YES
1	Light Control reserve	Not reached	Error
2	CPU Watchdog	OK	Reset generated
3	Operating Time Limit Warning	NO	YES
4	Battery charge	OK	Too low
5	Reference Point	Reached	Not reached
6	Minimum Velocity Flag	OK	Fall below
7	Maximum Velocity Flag	OK	Exceeded
8	Minimum Acceleration Flag	OK	Fall below
9	Maximum Acceleration Flag	OK	Exceeded
10	Position Limits Exceeded	OK	Exceeded
11-12	Reserved by CIP	Always 0	
13-15	Vendor specific		

**5-23.4.38 Supported Warnings – Attribute 48**

This attribute contains information on supported warnings by the position sensor device. This attribute is required when the *Warnings* attribute is implemented.

**Table 5-23.21 Supported Warnings Attribute Bit Definitions**

<b>Bit</b>	<b>Description</b>	<b>FALSE (0)</b>	<b>TRUE (1)</b>
0	Frequency Exceeded	Not supported	Supported
1	Light Control reserve	Not supported	Supported
2	CPU Watchdog	Not supported	Supported
3	Operating Time Limit Warning	Not supported	Supported
4	Battery charge	Not supported	Supported
5	Reference Point	Not supported	Supported
6	Minimum Velocity Flag	Not supported	Supported
7	Maximum Velocity Flag	Not supported	Supported
8	Minimum Acceleration Flag	Not supported	Supported
9	Maximum Acceleration Flag	Not supported	Supported
10	Position Limits Exceeded	OK	Exceeded
11-12	Reserved by CIP	Always 0	
13-15	Vendor specific		

**5-23.4.39 Warning flag – Attribute 49**

Indicates that warning error has occurred. This attribute is the logical OR of all the warnings bits in the *Warnings* attribute (attribute 47). This attribute is required when the *Warnings* attribute is implemented.

**5-23.4.40 Operating Time – Attribute 50**

This attribute is incremented as long as the encoder is powered. The *Operating Time* value is presented in tenths (0.1) of an hour.

**5-23.4.41 Offset Value – Attribute 51**

This attribute is required when the *Preset Value* (attribute 19) is implemented.

**Position Sensor Object, Class Code: 23<sub>Hex</sub>**

The *Offset Value* attribute is calculated by the preset function and shifts the *Position Value* attribute with the calculated value. The *Offset Value* is stored automatically by the device and can be read from the encoder for diagnostic purposes.

$$\text{Offset Value} \text{ (attribute 49)} = \text{Preset Value} \text{ (attribute 19)} - \text{Position Value} \text{ (attribute 10)}.$$

## 5-23.5 Common Services

The Position Sensor Object provides the following Common Services:

**Table 5-23.22 Position Sensor Object Common Service**

Service Code	Need In Implementation		Service Name	Description of Service
	Class	Instance		
0x05	Optional	NA	Reset	Resets all parameter values to the factory default
0x0D	Optional	NA	Apply_Attributes	Cause the configuration to become active
0x0E	Conditional *	Required	Get_Attribute_Single	Returns the contents of the specified attribute
0x10	N/A	Optional	Set_Attribute_Single	Modifies an attribute value
0x15	Optional	NA	Restore	Restores all parameter values from non-volatile storage
0x16	Optional	NA	Save	Saves all parameters to non-volatile storage
0x18	N/A	Conditional **	Get_Member	Returns an element of an array for specified attribute. Specifically intended for usage on "CAM" attributes.
0x19	N/A	Conditional **	Set_Member	Modifies an element of an array for specified attribute. Specifically intended for usage on "CAM" attributes.

\* The Get\_Attribute\_Single service is *required* at the class level if any class attributes are implemented

\*\* The Get\_Member and Set\_Member services are required when any of the array attributes are implemented.

See Appendix A for definition of these services

If several parameters will be modified resulting in side effects the Apply\_Attributes service shall be implemented to cause the configuration to become active after all parameters have been set. The Apply\_Attributes service shall validate parameter settings. If any parameters are in conflict, the error response Invalid value (0x09) shall be returned with the extended error code indicating the conflicting attribute Id.

**If Save or Restore service is implemented, all implemented attributes of the Position Sensor Object must be restored or saved to non-volatile storage.**

The reset service has the following parameter:

**Table 5-23.23 Reset Service**

Name	Type	Description of Request Parameters	Semantics of Values
Type	USINT	Type of Reset	See Table below

The parameter Type for the Reset service has the following bit specifications:

**Table 5-23.24 Reset Service Parameter Values**

Value	Type of Reset
0	Emulate as closely as possible cycling power. This value is the default if this parameter is omitted.
1	Return as closely as possible to the out-of-box configuration, then emulate cycling power as closely as possible.

## 5-23.6 Object-specific Services

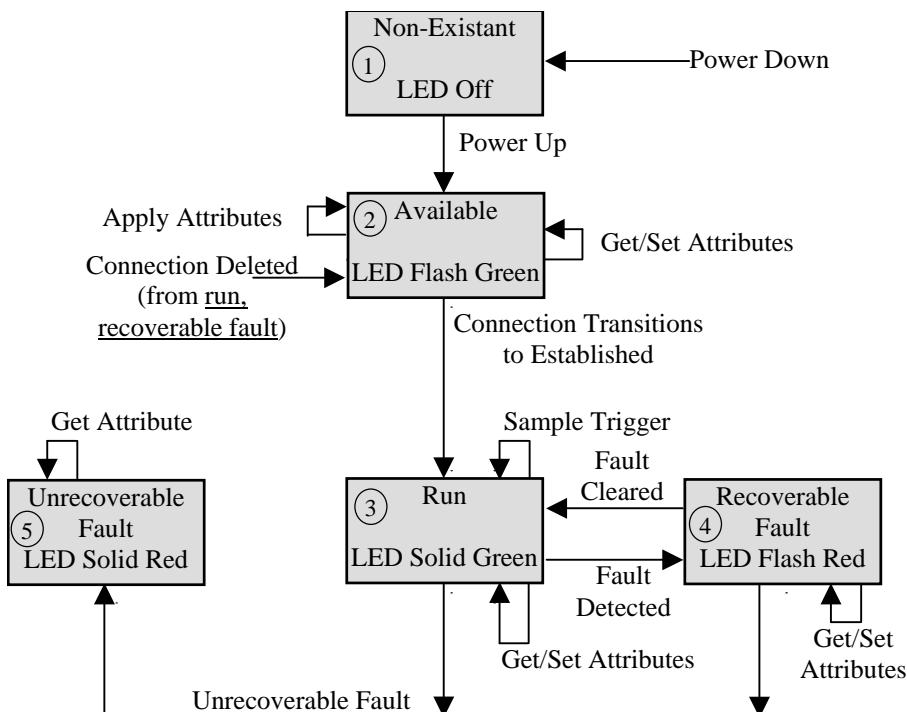
The Position Sensor Object provides no Object-specific services.

## 5-23.7 Behavior

The State Transition Diagram (Figure 5-23.27) provides a graphical description of the events and corresponding state transitions. A subset of the states and events may be supported in an application, but the behavior must be consistent.

The State Event Matrix (See Table 5-23.28) lists all pertinent events and the corresponding action to be taken while in each state.

**Important:** Events can occur simultaneously, but the *Fault* events have priority if they occur simultaneously with other events.

**Figure 5-23.25 State Transition Diagram for Position Sensor Object**

LED = I/O Status LED

Position Sensor Object, Class Code: 23<sub>Hex</sub>

**Important:** Events can occur simultaneously, but the *Fault* events have priority if they occur simultaneously with other events.

The following SEM contains these states:

- **Non-Existent:** a module without power.
- **Available:** waiting for a connection, power-up discrete input point defaults are set.
- **Run:** Position Sensor sensing data from its input and transmitting the data.
- **Recoverable Fault:** a recoverable fault has occurred.
- **Unrecoverable Fault:** an unrecoverable fault has occurred.

The SEM also contains these events:

**Table 5-23.26 Position Sensor Object Event Definitions**

This event	Is
Sample Trigger	a change of state, cyclic timer trigger, application trigger
Connection Deleted	I/O connection deleted.
Apply_Attributes	the Apply service of the I/O connection object the Position Sensor Object is connected to. Note: the application is responsible for validating the connection object's attributes.
Fault Cleared	the application clearing a detected fault
Connection Transitions to Established	I/O connection transitions to Established.
Connection Transitions to Timed Out state	I/O connection transitions to Timed-Out

The figure below is a **conceptual** illustration of the state machine for a typical input object (producing application). The events listed above are represented by the dotted line labeled “state change.”

**Figure 5-23.27 Conceptual Position Sensor Object Operation**

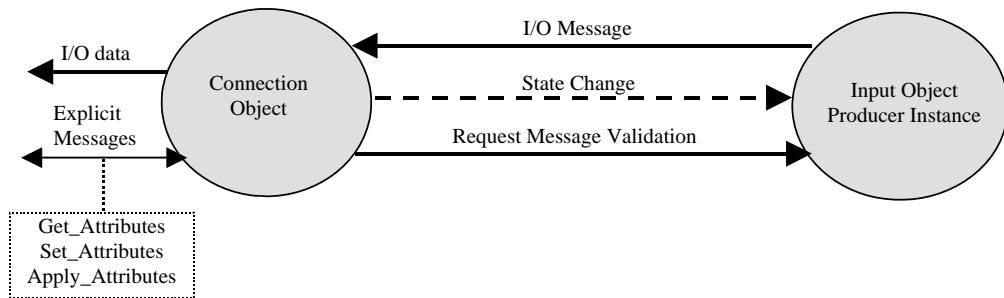


Table 5-23.28 State Event Matrix for the Position Sensor Object

Event	State				
	Non-Existent	Available	Run	Recoverable Fault	Unrecoverable Fault
Sample Trigger	Not Applicable	Ignore event	Sample data, Send data	Ignore event	Ignore event
Apply Attributes	Not Applicable	Verify attributes, return result	Return error (Object State Conflict)	Return error (Object State Conflict)	Ignore event
Connection Deleted	Not Applicable	Ignore Event	Transition to Available	Transition to Available	Ignore event
Connection Transitions to Established	Not Applicable	Transition to Run	Ignore event	Ignore event	Ignore event
Connection Transitions to Timed Out state	Not Applicable	Ignore event	Transition to Recoverable Fault	Ignore event	Ignore event
Fault Cleared	Not Applicable	Not Applicable	Not Applicable	Transition to Run	Ignore event
Get_Attribute	Return Error (Object Does Not Exist)	Return value	Return value	Return value	Return Value
Set_Attribute	Return Error (Object Does Not Exist)	Accept value	Accept value	Accept value	Ignore event
I/O Status LED	Off	Off	Solid Green	Flash Red	Solid Red
Reset	Not Applicable	Perform service	Perform service	Attempt to Recover	Ignore Event
Restore	Not Applicable	Restore from storage	Restore from storage	Ignore Event	Ignore Event
Save	Not Applicable	Stores parameter to non-volatile memory	Stores parameter to non-volatile memory	Ignore Event	Ignore Event

## 5-24 Position Controller Supervisor Object

**Class Code: 24 hex**

The position controller supervisor handles errors for the position controller as well as Home and Registration inputs.

### 5-24.1 Revision History

Since the initial release of this object class definition, changes have been made that require a revision update of this object class. The table below represents the revision history:

**Table 5-24.1 Position Controller Supervisor Object Revision History**

Revision	Description
01	Initial release
02	Class Attributes 32 and 33 added

### 5-24.2 Class Attributes

**Table 5-24.2 Position Controller Supervisor Object Class Attributes**

Attr ID	Need in Implem	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1	Required	Get	Revision	UINT	Revision of this object.	The current value assigned to this attribute is two (02).
2 thru 7	These class attributes are either optional or conditional and are described in Chapter 4 of this specification.					
32	Required	Get	Consumed Axis Selection Number	USINT	Specifies the axis number to which the data contained in the I/O Command Message is routed.	Value in the range of 1-7
33	Required	Get	Produced Axis Selection Number	USINT	Specifies the axis number to which the data contained in the I/O Response Message is routed.	Value in the range of 1-7

#### 5-24.2.1 Consumed Axis Selection Number

When an I/O Message is consumed by the Position Controller Supervisor object, its internal destination may vary. The destination of the I/O Message shall be specified within the *Command Axis Number*. See the description of I/O Connection Messages of the Position Controller Device Profile in Chapter 6 for details.

For static systems, this attribute is normally specified in the *Consumed Connection Path* of the respective I/O Connection instance.

### 5-24.2.2 Produced Axis Selection Number

When an I/O Message is produced by the Position Controller Supervisor object, its internal source may vary. The source of the I/O Message shall be specified within the *Response Axis Number*. See the description of I/O Connection Messages of the Position Controller Device Profile in Chapter 6 for details.

For static systems, this attribute is normally specified in the *Produced Connection Path* of the respective I/O Connection instance.

## 5-24.3 Object Instance Attributes

### 5-24.3.1 Supervisor Attributes

**Table 5-24.3 Position Controller Supervisor Instance Attributes – Supervisor Attributes**

Attr ID	Need in Implem	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1	Optional	Get	Number of Attributes	USINT	The total number of attributes supported by this object in this device	Return value is in the range of 0 to 255.
2	Optional	Get	Attribute List	Array of USINT	Returns an array with a list of the attributes supported by this object in this device.	Array size defined by attribute 1.
3	Required	Get	Axis Number	USINT	Returns the axis number which is the same as the instance for this object.	This value will be in the range of 1 to 7. The axis number is the same as the instance number for all of the axis objects: position controller supervisor, position controller, drive, motor data, and block sequencer.
4			Reserved			
5	Required	Get	General Fault	BOOL	General Fault flag. This bit is logical OR of all fault condition attribute flags in the device. This bit is reset when the fault condition is removed.	1 = fault condition exists.
6	Required	Set	Command Message Type	USINT	Sets the command message type that is being sent by the controlling device.	Valid Message Type codes are 1 to 1F hex. 1 = Type 01 hex 2 = Type 02 hex, etc.
7	Required	Set	Response Message Type	USINT	Sets the response message that is returned to the controlling device	Valid Message Type codes are 1 to 1F hex.. 1 = Type 01 hex 2 = Type 02 hex etc.
8	Optional	Get	Fault Input	BOOL	Fault input fault	1 = fault input is active.

**Position Controller Supervisor Object, Class Code: 24<sub>Hex</sub>**

Attr ID	Need in Implem	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
9	Optional	Set	Fault Input Action	USINT	Action taken when fault input becomes active code.	0 = Command Output Generator Off, 1 = Hard Stop, 2 = smooth stop, 3 = no action. 4 - 127 = Reserved by CIP 128 - 255 = Vendor specific

### 5-24.3.2 Home and Index Attributes

**Table 5-24.4 Position Controller Supervisor Instance Attributes – Home and Index Attributes**

Attr ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
10	Optional	Set	Home Action	USINT	Home input action code. Action taken when armed home input is triggered.	0 = Command Output Generator off, 1 = Hard stop, 2 = Smooth stop, 3 = No action, 4 = Gate index. 5 - 127 = Reserved by CIP 128 - 255 = Vendor specific
11	Optional	Set	Home Active level	BOOL	Home trigger Active Level flag is used to program the Home inputs active level.	0 = active low, 1 = active high.
12	Optional	Get/Set	Home Arm	BOOL	Home trigger arm flag is used to arm the Home input.	1 arms the home input, reading a 0 indicates the trigger has occurred.
13	Optional	Set	Index Action	USINT	Index input action code.	0 = Command Output Generator off, 1 = Hard stop and, 2 = Smooth stop, 3 = No action. 4 - 127 = Reserved by CIP 128 - 255 = Vendor specific
14	Optional	Get/Set	Index Active Level	BOOL	Used to program the Index inputs active level.	0 = active low, 1 = active high.
15	Optional	Get/Set	Index Arm	BOOL	Index trigger arm flag is used to arm the Index input.	1 arms the index input, reading 0 indicates the trigger has occurred.
16	Optional	Get	Home Input Level	BOOL	Actual level of the Home input.	0 = Home input is low 1 = Home input is high.
17	Optional	Get	Home Position	DINT	Home trigger position reflects the position at the time the home input is triggered.	This value can be in the range of 0x80000001 to 0x7FFFFFFF.

**Position Controller Supervisor Object, Class Code: 24<sub>Hex</sub>**

Attr ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
18	Optional	Get	Index Position	DINT	Index trigger position reflects the position at the time the home input is triggered.	This value can be in the range of 0x80000001 to 0x7FFFFFF.

**5-24.3.3 Registration Attributes****Table 5-24.5 Position Controller Supervisor Instance Attributes – Registration Attributes**

Attr ID	Need in Implem	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
19	Optional	Set	Registration Action	USINT	Registration input action defines what happens when the registration input is triggered.	0 = Command Output Generator off 1= Hard Stop 2 = Smooth stop 3 = No action. 4 = Go to Reg position offset 5 = Go to Reg position absolute. 6 - 127 = Reserved by CIP 128 - 255 = Vendor specific
20	Optional	Set	Registration Active level	BOOL	Registration trigger Active Level flag is used to program the Registration inputs active level.	0 = active low, 1 = active high.
21	Optional	Get/Set	Registration Arm	BOOL	Registration trigger arm flag is used to arm the Registration input.	Set to 1 to arm the registration input, reading a 0 indicates the registration trigger has occurred.
22	Optional	Get	Registration Input Level	BOOL	Actual level of the registration input.	0 = registration is low 1 = registration is high.
23	Optional	Set	Registration Offset	DINT	Defines a position value that is used as an offset or absolute position dependent on the registration action code.	This value can be in the range of 0x80000001 to 0x7FFFFFF.
24	Optional	Get	Registration Position	DINT	Registration trigger position reflects the position at the time the Registration input is triggered.	This value can be in the range of 0x80000001 to 0x7FFFFFF.

**5-24.3.4 Axis Following Attributes****Table 5-24.6 Position Controller Supervisor Instance Attributes – Axis Following Attributes**

Attr ID	Need in Implem	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
25	Optional	Set	Follow Enable	BOOL	Follow Enable enables following of the Follow Axis.	0 = following disabled, 1 = following enabled.
26	Optional	Set	Follow Axis	USINT	Specifies the Axis to follow.	0 = no following, 1 to 255 specifies the axis.
27	Optional	Set	Follow Divisor	DINT	Used to calculate the Command Position by dividing the Follow Axis position with this value.	
28	Optional	Set	Follow Multiplier	DINT	Used to calculate the Command Position by multiplying the Follow Axis position with this value.	

**5-24.4 Common Services****Table 5-24.7 Position Controller Supervisor Common Services**

Need in Implementation	Name	Service Code	Description of Service
Required	Get_Attribute_Single	0Ehex	Returns the contents of the specified attribute
Required	Set_Attribute_Single	10hex	Modifies the attribute value.

See Appendix A for definitions of these services

**5-24.5 Object-specific Services**

The Position Controller Supervisor object provides no object-specific services.

**5-24.6 Behavior**

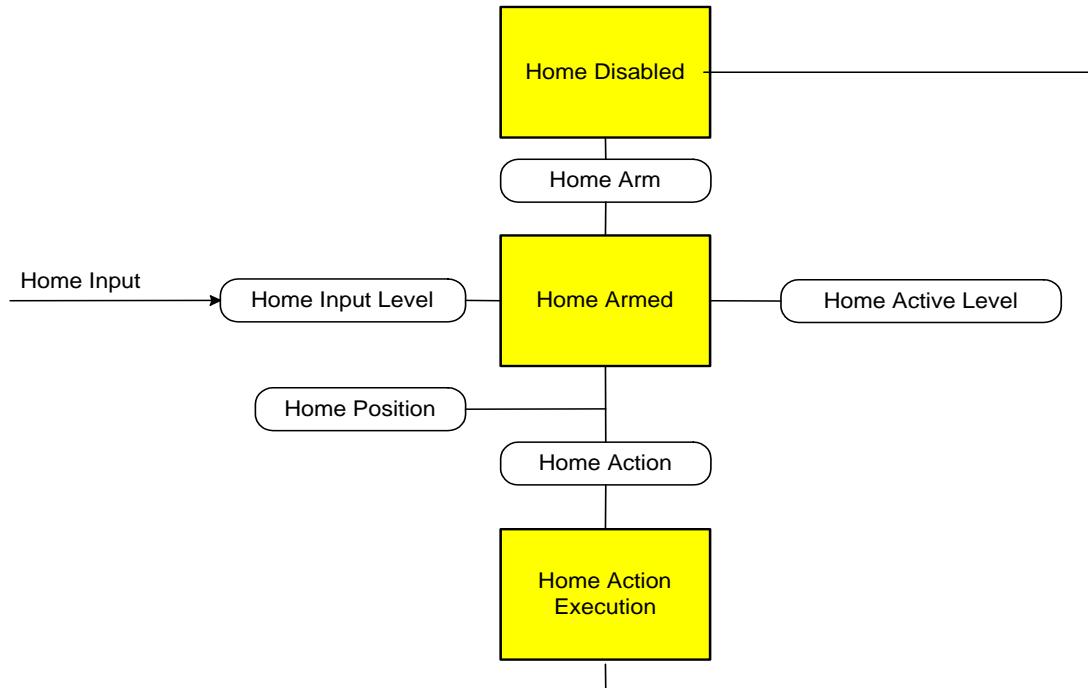
Object behavior is defined by the following state diagrams.

### 5-24.6.1 Position Controller Supervisor State Diagrams

#### 5-24.6.1.1 Home Input State Diagrams

The following state diagram describes the behavior of the Home input. The Home input active level can be programmed. Home will trigger when it is armed and the input goes to the active level. When home is triggered the home action is performed and the home input returns to the disabled state.

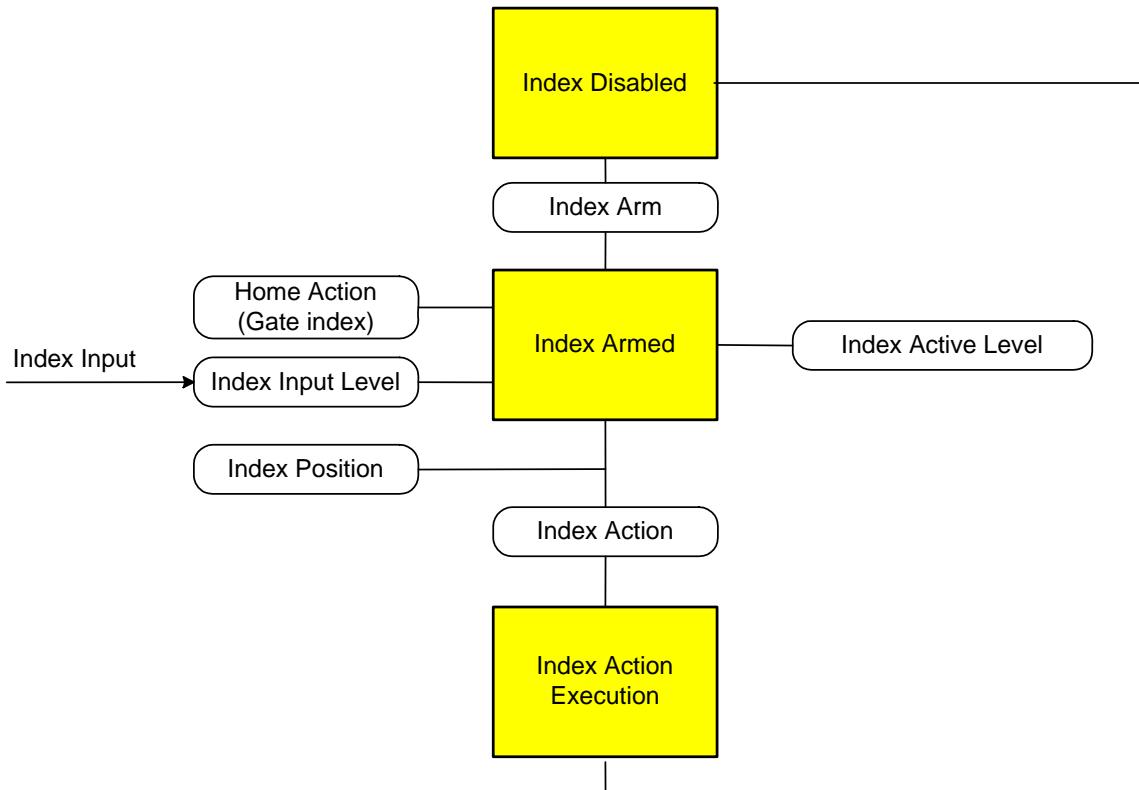
Figure 5-24.8 Home Input State Diagram



### 5-24.6.1.2 Index Input State Diagram

The following diagram describes the behavior of the Index input. The Index input active level can be programmed. Index will trigger when it is armed and the input goes to the active level. In addition the index input can be gated using the Home input when the Home Action attribute is set to Gate index. When index is triggered the index action is performed and the Index input returns to the disabled state.

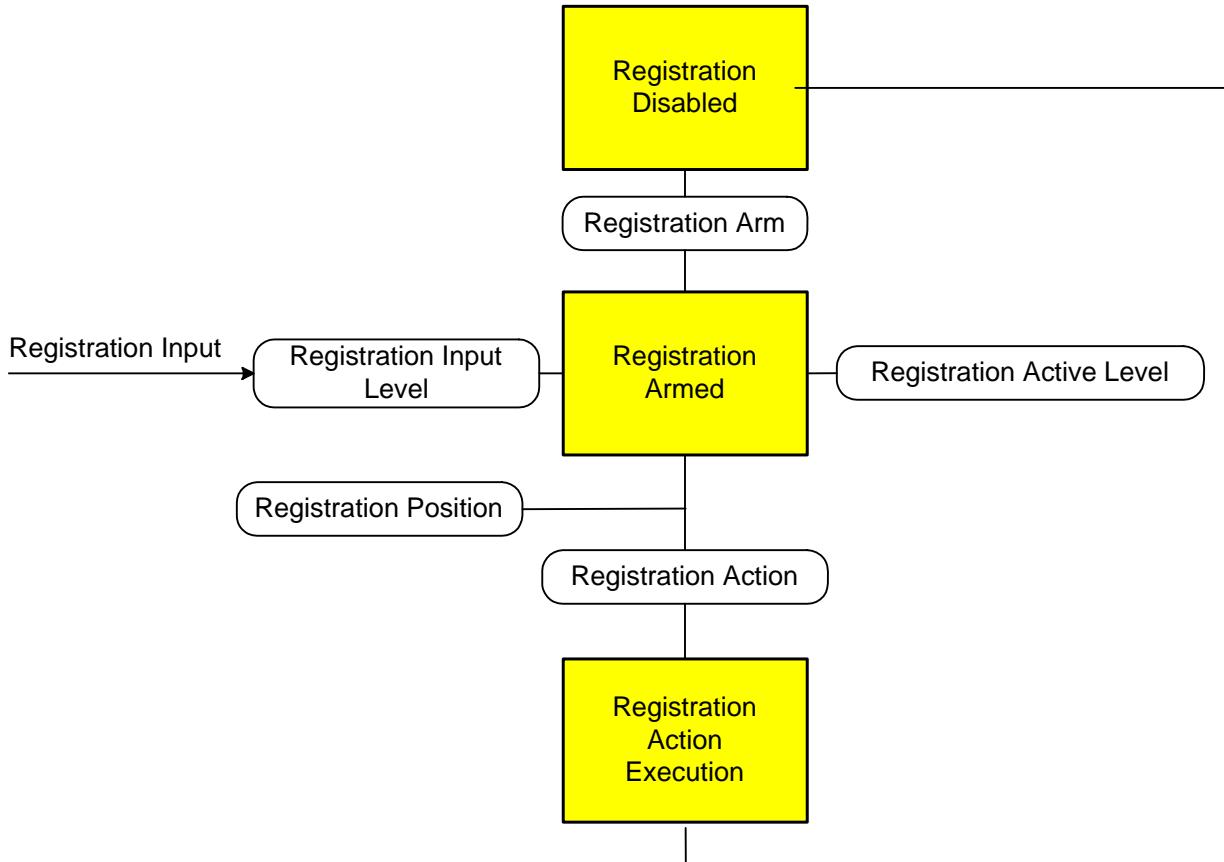
Figure 5-24.9 Index Input State Diagram



### 5-24.6.1.3 Registration Input State Diagram

The following state diagram describes the behavior of the Registration input. The Registration input active level can be programmed. The Registration input will trigger when it is armed and the input goes to the active level. When the Registration input is triggered the Registration action is performed and the Registration input returns to the disabled state.

Figure 5-24.10 Registration Input State Diagram



## **5-25 Position Controller Object**

**Class Code: 25<sub>hex</sub>**

The position controller object performs the control output velocity profiling and handles input and output to and from the motor drive unit, limit switches registration etc.

### **5-25.1 Revision History**

Since the initial release of this object class definition, changes have been made that require a revision update of this object class. The table below represents the revision history:

**Table 5-25.1 Position Controller Object Revision History**

Revision	Description
01	Initial release
02	Instance Attribute 58 added

### **5-25.2 Class Attributes**

**Table 5-25.2 Position Controller Object Class Attributes**

Number	Need in Implem	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1	Required	Get	Revision	UINT	Revision of this object.	The current value assigned to this attribute is two (02).
2 thru 7	These class attributes are either optional or conditional and are described in Chapter 4 of this specification.					

### 5-25.3 Instance Attributes

**Table 5-25.3 Position Controller Object Instance Attributes**

Attr ID	Need in Implem	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1	Required	Get	Number of Attributes	USINT	Returns the total number of attributes supported by this object in this device.	Return value is in the range of 0 to 255.
2	Required	Get	Attribute List	Array of USINT	Returns an array with a list of the attributes supported by this object in this device.	Array size defined by attribute 1.
3	Optional	Set	Mode	USINT	Operating mode.	0 = Position mode(default), 1 = Velocity mode, 2 = Torque mode.
4	Optional	Set	Position Units	DINT	Position Units ratio value is the number of actual position feedback counts equal to one position unit.	Set this value to a positive number only. Default = 1.
5	Optional	Set	Profile Units	DINT	Profile Units ratio value is the number of actual position feedback counts per second or second <sup>2</sup> equal to one velocity, acceleration or deceleration unit..	This value is set to a positive number only. Default = 1.
6	Required	Set	Target Position	DINT	Profile move position defined in position Units	This value can be in the range of 0x80000001 to 0x7FFFFFF.
7	Required	Set	Target Velocity	DINT	Profile velocity defined in profile units per second.	This value is set to a positive number only.
8	Required	Set	Acceleration	DINT	Profile Acceleration rate defined in profile units per second <sup>2</sup> .	This value is set to a positive number only.
9	Optional	Set	Deceleration	DINT	Profile Deceleration rate defined in profile units per second <sup>2</sup> .	This value is set to a positive number only.
10	Optional	Set	Incremental Position Flag	BOOL	Incremental Position Flag	If set to 0 the target position (attribute 6) will be interpreted as absolute. If set to 1 the target position will be interpreted as incremental
11	Required	Set	Load Data/Profile Handshake	BOOL	Used to Load Command Data, Start a Profile Move, and indicate that a Profile Move is in progress.	See “Semantics” at end of this table.
12	Optional	Get	On Target Position	BOOL	On target position flag indicates that the motor is within the deadband (Attribute 38) distance to the target position.	Reads Set when the Target position equals the actual position within deadband limits. Will clear if target position is changed.

Position Controller Object, Class Code: 25<sub>Hex</sub>

Attr ID	Need in Implem	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
13	Optional	Set	Actual Position	DINT	Actual Absolute position value equals the real position in position units. Set to re-define Actual Position	When set this value can be in the range of 0x80000001 to 0x7FFFFFFF.
14	Optional	Get	Actual Velocity	DINT	Actual Velocity in profile units/sec.	Value read will be positive.
15	Optional	Get	Commanded Position	DINT	This value equals the instantaneous calculated position.	When read this value will be in the range of 0x80000001 to 0x7FFFFFFF.
16	Optional	Get	Commanded Velocity	DINT	This value equals the Instantaneous calculated velocity in profile units per second.	Value read will be positive.
17	Optional	Set	Enable	BOOL	Enable Output.	Set to enable drive and feedback, clear to disable.
18	Optional	Set	Profile Type	USINT	Profile Type code defines the type of move profile.	0 = Trapazoidal, 1 = S-Curve, 2 = Parabolic. 3 - 127 = Reserved by CIP 128 - 255 = Vendor specific
19	Optional	Set	Profile Gain	DINT	This attribute provides a gain value for non-trapezoidal profiles such as S-Curve profiling. The implementation and function of this gain is vendor specific.	Range is defined by the vendor.
20	Optional	Set	Smooth Stop	BOOL	Smooth stop motor.	Set to force immediate deceleration to zero velocity at programmed decel rate.
21	Optional	Set	Hard Stop	BOOL	Hard stop motor.	Set to force immediate deceleration to zero velocity at max decel rate.
22	Optional	Set	Jog Velocity	DINT	Defines the jogging velocity in profile units per second.	This value is set to a positive number only.
23	Optional	Set	Direction	BOOL	Instantaneous direction	0 = negative or reverse direction and 1 = positive or forward. This value can be set in Velocity mode to change direction.
24	Optional	Set	Reference Direction	BOOL	Defines direction.	0 = forward direction is clockwise, 1 = reverse direction is counter clockwise as viewed from the load end of the motor shaft.
25	Optional	Set	Torque	DINT	Output torque.	Set this value to change the torque (Torque mode only) or read the current torque. 0 = no torque output. Range defined by the vendor.

Position Controller Object, Class Code: 25<sub>Hex</sub>

Attr ID	Need in Implem	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
26	Optional	Set	Positive Torque Limit	DINT	This value sets the maximum allowable torque output in the positive direction.	This value is set to a positive number only. Range defined by the vendor.
27	Optional	Set	Negative Torque Limit	DINT	This value sets the maximum allowable torque output in the negative direction.	This value is set to a negative number only. Range defined by the vendor.
28			Reserved			
29	Optional	Get	Wrap Around	BOOL	Position Wrap Around indicator Flag	If set to 1 the motor has gone past its maximum position. This can only happen in Velocity mode and is not necessarily a fault condition. This bit is reset when read.
30	Optional	Set	Kp	INT	Proportional gain.	Range is 0 to 32767.
31	Optional	Set	Ki	INT	Integral gain.	Range is 0 to 32767.
32	Optional	Set	Kd	INT	Derivative gain.	Range is 0 to 32767.
33	Optional	Set	MaxKi	INT	Integration limit.	Range is 0 to 32767.
34	Optional	Set	KiMode	USINT	Integration limit Mode.	0 = use Ki term at all times, 1 = use Ki term only when stopped and holding position.
35	Optional	Set	Velocity Feed Forward.	INT	Velocity feed forward gain value.	Range is 0 to 32767.
36	Optional	Set	Accel Feed Forward	INT	Acceleration feed forward gain value.	Range is 0 to 32767.
37	Optional	Get	Sample Rate	INT	Update sample rate in $\mu$ Seconds.	Value returned is positive.
38	Optional	Set	Position Deadband	USINT	Set this value to prevent axis hunting within the desired window.	Range is 0 to 255.
39	Optional	Set	Feedback Enable	BOOL	This flag will set or clear automatically with the Enable attribute (17). Feedback can be turned off, using this attribute, leaving the enable on, for offset adjustments on the drive unit	0 = command output generator off, 1 = command output generator on.
40	Optional	Set	Feedback Resolution	DINT	Feedback resolution in counts. Number of actual position feedback counts in one revolution of the position feedback device.	This value is set to a positive number only.
41	Optional	Set	Motor Resolution	DINT	Motor resolution in motor steps. Number of motor steps in one revolution of the motor.	This value is set to a positive number only.

Position Controller Object, Class Code: 25<sub>Hex</sub>

Attr ID	Need in Implem	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
42	Optional	Set	Position Tracking Gain	DINT	Position Tracking Gain (Stepper) Gain value for position maintenance to control steppers with position feedback.	Range defined by the vendor.
43	Optional	Set	Max Correction velocity	UINT	Position maintenance value to prevent stepper motor stalls. Value in counts per second.	This value is set to a positive number only.
44	Optional	Set	Max Static Following Error	DINT	Maximum allowable following error when the motor is stopped and holding position. If the difference between actual and commanded position exceeds this value, following error flag is set.	Set to value to a positive number only.
45	Optional	Set	Max Dynamic Following Error	DINT	Maximum allowable following error when the motor is in motion. If difference between actual and commanded position exceeds this value, following error flag is set.	Set to value to a positive number only.
46	Optional	Set	Following Error Action	USINT	Following error action code.	0 = Command Output Generator Off, 1 = Hard Stop, 2 = Smooth stop 3 = no action. 4 - 127 = Reserved by CIP 128 - 255 = Vendor specific
47	Optional	Set	Following Error Fault	BOOL	Following error occurrence flag. Set when a following error occurs. This bit is reset when another move is attempted.	Set when a following error occurs. This bit can be cleared directly or by re-programming the Following Error Action attribute.
48	Optional	Get	Actual Following Error	DINT	Actual Following Error.	This value is the actual amount of Following Error in position feedback counts.
49	Optional	Set	Hard Limit Action	USINT	Hard limit action code.	0 = Command Output Generator Off 1 = Hard Stop 2 = smooth stop. 3 - 127 = Reserved by CIP 128 - 255 = Vendor specific
50	Optional	Get	Forward Limit	BOOL	Motion is not allowed in the positive direction when active.	Set when the forward limit stop is active.
51	Optional	Get	Reverse Limit	BOOL	Motion is not allowed in the negative direction when active.	Set when a reverse limit stop is active.
52	Optional	Set	Soft Limit Enable	BOOL	Enables soft limits	When set, motion that exceeds the defined limits will result in a motor stop.

Position Controller Object, Class Code: 25<sub>Hex</sub>

Attr ID	Need in Implem	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
53	Optional	Set	Soft Limit Action	USINT	Soft limit action code.	0 = Command Output Generator Off, 1 = Hard Stop, 2 = smooth stop. Action codes 128 through 255 are for vendor specific action.
54	Optional	Set	Positive Soft Limit Position	DINT	Soft limit positive boundary defined in position units..	This value can be in the range of 0x80000001 to 0x7FFFFFFF.
55	Optional	Set	Negative Soft Limit Position	DINT	Soft limit negative boundary defined in position units..	This value can be in the range of 0x80000001 to 0x7FFFFFFF.
56	Optional	Get	Positive Limit Triggered	BOOL	Hard Forward limit occurrence flag.	Set when a positive limit stop occurs.
57	Optional	Get	Negative Limit Triggered	BOOL	Hard Reverse limit occurrence flag.	Set when a negative limit stop occurs.
58	Required	Get	Load Data Complete	BOOL	Indicates that valid data for a valid I/O command message type has been loaded into the position controller device.	See "Semantics" at the end of this table.

**5-25.3.1 Semantics****5-25.3.1.1 Profile in Progress**

This attribute performs three functions:

- loading data in the I/O command message;
- starting a Profile Move in the both the i/o command message and explicit messaging; and
- indicating if a Profile Move is in process. See the chart below for a complete explanation of the device's behavior with respect to this bit.

**Position Controller Object, Class Code: 25<sub>Hex</sub>****Table 5-25.4 Behavior of Device With Respect to Profile In Progress Attribute**

Connection	Message Type or Service	Bit Name	Behavior
I/O	Command	Load Data/Start Profile	When this bit transitions from zero to one, the position controller device will attempt to load the data contained in the command message data bytes. If the command message type contained in the command message is the command message type that starts a Profile Move, the Profile Move will start. See the table in Section 6-18.4.3 for an explanation of which command message type starts a Profile Move for each mode.
	Response	Profile in Progress	This bit will indicate that a Profile Move is in progress. This bit will read 1 when a Profile Move is started and will remain 1 until the Profile Move is complete or terminated, at which point it will be zero.
Explicit	Set_Attribute_Single	Start Profile	A "Set_Attribute_Single" service, which sets this attribute to 1 will start a Profile Move. A "Set_Attribute_Single" service, which sets this attribute to 0 will have no effect.
	Get_Attribute_Single	Profile in Progress	This bit will indicate that a Profile Move is in progress. This bit will be one after receipt of a set attribute service which sets this attribute to 1 and will remain 1 until the profile move is complete or terminated, at which point it will be zero.

**5-25.3.1.2 Load Data Complete**

This bit is used for data handshaking in the I/O message. It reflects that the position controller device has successfully loaded the data contained in the I/O command message data bytes. This attribute will be reset when the Load Command Data/Start Profile bit is reset. Refer to the Position Controller Device Profile in Chapter 6 for an explanation of the handshaking procedure. This bit is not affected by explicit messaging.

**5-25.4 Common Services****Table 5-25.5 Position Controller Object Common Services**

Need in Implementation	Name	Service Code	Description of Service
Required	Get_Attribute_Single	0Ehex	Returns the contents of the specified attribute
Required	Set_Attribute_Single	10hex	Modifies the attribute value.

See Appendix A for definition of these services

**5-25.5 Object-specific Services**

The Position Controller object provides no object-specific services.

**5-25.6 Behavior**

Object behavior is defined by the following state diagrams

## 5-25.6.1 Position Controller State Diagrams

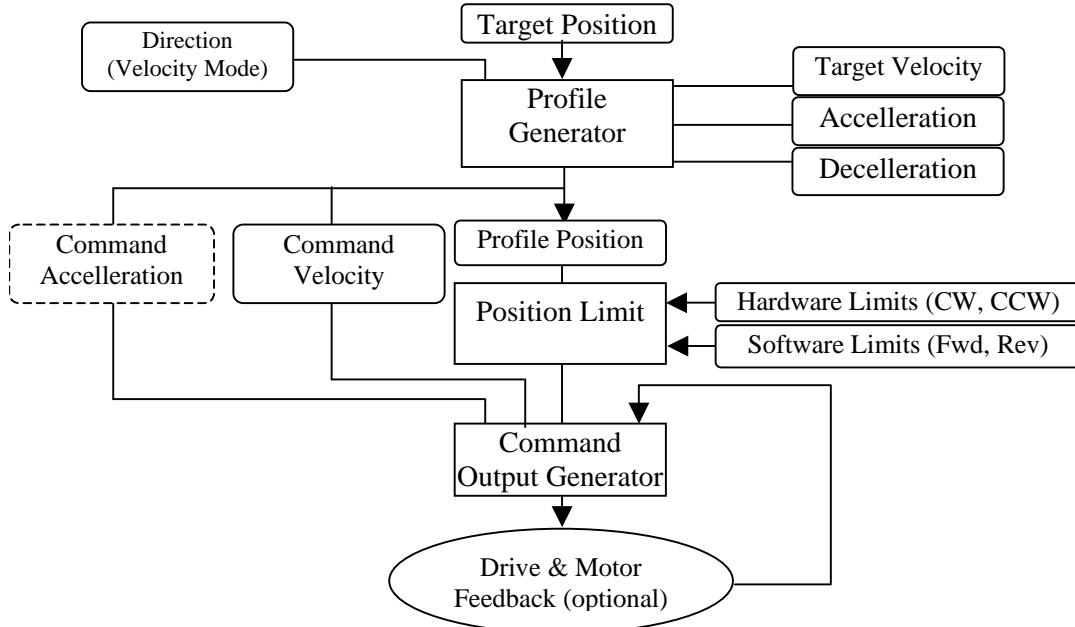
### 5-25.6.1.1 Profile Move Generation State Diagrams

The state diagram below describes Position Controller profile generation. The Profile generator uses Acceleration, Target, Velocity and Deceleration to perform a profile move to the Target Position. In profile velocity mode the Target Position is infinite with polarity defined by the direction attribute until such time the device is commanded to decelerate and stop.

After the profile position is generated, it passes through a limit filter. If the generated Profile position is outside the defined software limits, or if a hardware limit is active, the profile is modified and the output Command position is limited. The limit function and attributes are optional.

The Command Position is then sent to the output generator, which produces a control signal. The output generator can be servo, stepper or some other method for controlling position. Commanded Acceleration and Velocity are also sent to the output generator for feed forward purposes. Feedback is optional unless required by the output method being used.

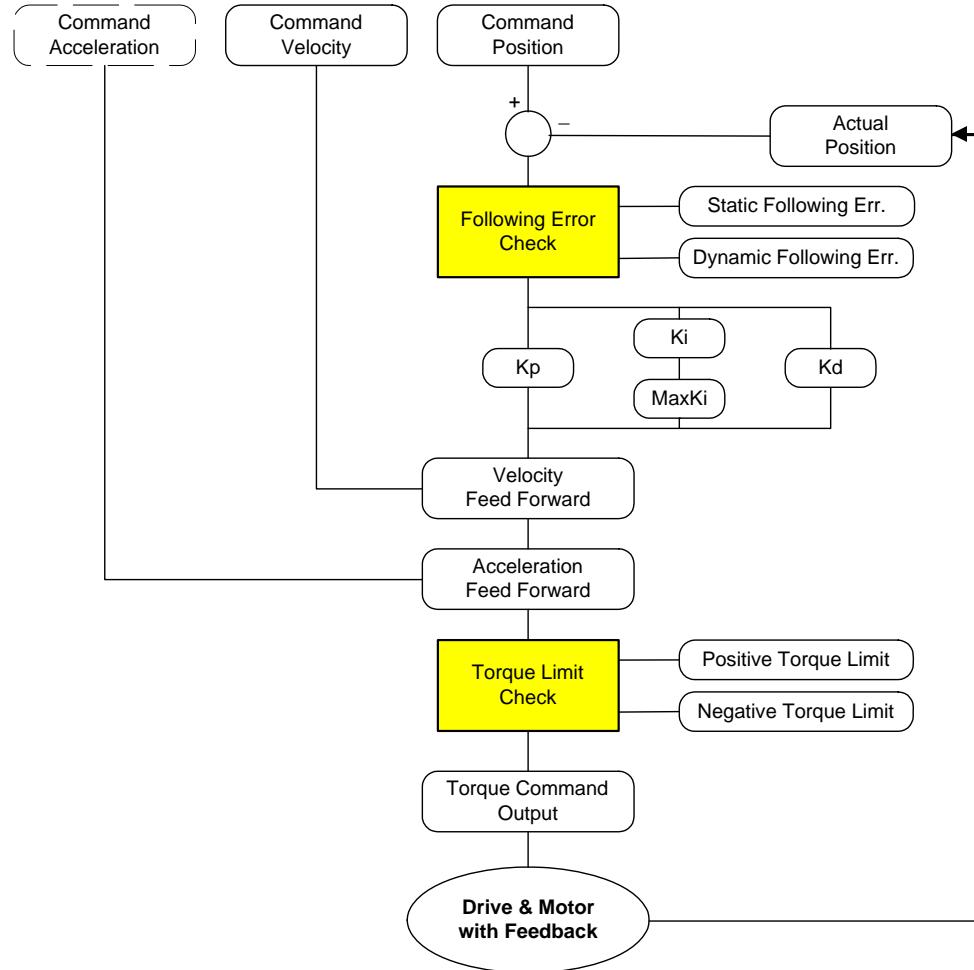
**Figure 5-25.6 Position Controller Profile Generation State Diagram**



### 5-25.6.1.2 Servo Output Generation State Diagram

The diagram below is an example of a Servo Command output generator. Commanded Position, Velocity and Acceleration input comes from the Profile Generation diagram.

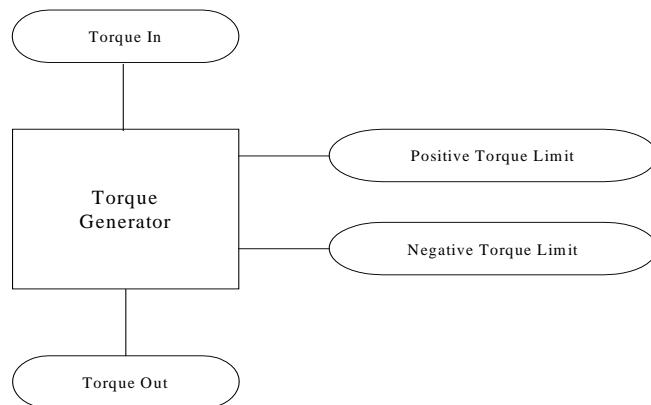
**Figure 5-25.7 Servo Output Generator State Diagram**



### **5-25.6.1.3    Torque Mode Output State Diagram**

The following diagram describes the direct torque mode function.

**Figure 5-25.8 Torque Mode State Diagram**



## 5-26 Block Sequencer Object

**Class Code: 26 hex**

This object handles the execution of Command Blocks or Command Block chains.

### 5-26.1 Class Attributes

**Table 5-26.1 Block Sequencer Object Class Attributes**

Number	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1 thru 7	These class attributes are optional and are described in Chapter 4 of this specification.					

### 5-26.2 Instance Attributes

These attributes can be used for control, configuration or status.

**Table 5-26.2 Block Sequencer Object Instance Attributes**

Attr ID	Need in Implem	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1	Required	Set/Get	Block	USINT	Instance number of starting Command Block.	This value Defines the Command Block instance to execute. Set from 1 to 255.
2	Required	Get/Set	Block Execute	BOOL	Block execution flag.	Setting this value executes the block defined by the Block Attribute (1). When this value reads back cleared, the block or chain of blocks is done.
3	Required	Get	Current Block	USINT	Current block in execution.	This value contains the Command Block instance number of the currently executing block (1 - 255).
4	Required	Get	Block Fault	BOOL	Block fault flag.	Set when a block error occurs, such as the Wait Equals command time-out or execution of an invalid Command Block. When a Block Fault occurs block execution will stop. This bit is reset when the Block Fault Code attribute (5) is read.
5	Optional	Get	Block Fault Code	USINT	Block fault Code.	Defines the specific block fault. 0 = no fault, 1 = invalid or empty block data, 2 = command time-out (Wait Equals), 3 = execution fault.
6	Optional	Set	Counter	DINT	Sequencing Counter.	Must be positive. Counter that can be used for sequencing loops.

### 5-26.3 Common Services

**Table 5-26.3 Block Sequencer Object Common Services**

Need in Implementation	Name	Service Code	Description of Service
Required	Get_Attribute_Single	0Ehex	Returns the contents of the specified attribute
Required	Set_Attribute_Single	10hex	Modifies the attribute value.

See Appendix A for a description of these services

### 5-26.4 Object-specific Services

The Block Sequencer object provides no object-specific services.

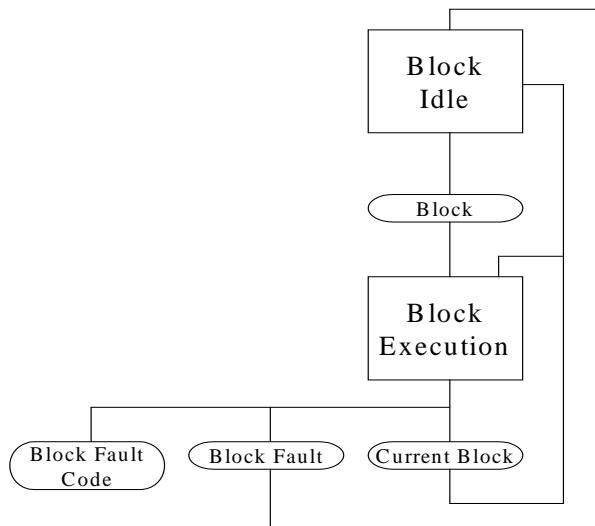
### 5-26.5 Behavior

Object behavior is defined by the following state diagrams

#### 5-26.5.1 Block Sequencer State Diagrams

The diagram below describes the Block Sequencer. When the Sequencer is commanded to execute a block, the Block Execution state is entered. Block execution continues on consecutive blocks until the end of the linked chain is reached or an error occurs.

**Figure 5-26.4 Block Sequencer State Diagram**



## 5-27 Command Block Object

**Class Code: 27 hex**

Each instance of the Command Block object defines a specific command. These blocks can be linked to other blocks to form a command block chain.

### 5-27.1 Class Attributes

**Table 5-27.1 Command Block Object Class Attributes**

Number	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1 thru 7	These class attributes are optional and are described in Chapter 4 of this specification.					

### 5-27.2 Attributes

These attributes are downloaded at configuration time and executed at run time through the Block Sequencer object. Attributes 3, 4, and 5 definitions are dependent on the block command attribute (01). Each instance of the Block command class defines a different block command that can be linked to any other block command instance to form an execution sequence.

**Table 5-27.2 Command Block Object Instance Attributes**

Attr ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1	Required	Set	Block Command	USINT	Block Command #	Defines the format of the block data. Command data formats are defined below.
2	Required	Set	Block Link #	USINT	Block link instance number.	This value provides a link to the next block instance to execute. When this block is done, the link block will be executed.
3	Depends on Command	Set	Depends on Command	Depends on Command		Refer to the command definitions for the description of attribute 3.
4	Depends on Command	Set	Depends on Command	Depends on Command		Refer to the command definitions for the description of attribute 4.
5	Depends on Command	Set	Depends on Command	Depends on Command		Refer to the command definitions for the description of attribute 5.
6	Depends on Command	Set	Depends on Command	Depends on Command		Refer to the command definitions for the description of attribute 6.
7	Depends on Command	Set	Depends on Command	Depends on Command		Refer to the command definitions for the description of attribute 6.

## 5-27.3 Command Specific Attribute Services

This section defines the Command block data. Command blocks can be linked together to form a command block chain. Looping and branching commands are supported.

### 5-27.3.1 Modify Attribute Command - 01

This command is used to change an attribute's value. Attribute 1 must be set to 01.

**Table 5-27.3 Command Block - Modify Attribute Command**

Attr ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
3	Required for Command 01	Set	Target Class	USINT	Target class number to perform block sequencing on.	This value Defines the class which will be sequenced.
4	Required for Command 01	Set	Target Instance	USINT	Target instance number of the class to perform block sequencing on.	This value Defines the instance of the class which will be sequenced.
5	Required for Command 01	Set	Attribute #	USINT	Position Controller Attribute	Position Controller class attribute number. Must be a settable attribute.
6	Required for Command 01	Set	Attribute Data	Dependent on attribute #	Attribute Data.	The new Attribute data.

### 5-27.3.2 Wait Equals Command - 02

This command is used to stop execution of a linked chain of command until an attribute becomes valid. Attribute 1 must be set to 02.

**Table 5-27.4 Command Block – Wait Equals Command**

Attr ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
3	Required for Command 02	Set	Target Class	USINT	Target class number to perform block sequencing on.	This value Defines the class which will be sequenced.
4	Required for Command 02	Set	Target Instance	USINT	Target instance number of the class to perform block sequencing on.	This value Defines the instance of the class which will be sequenced.
5	Required for Command 02	Set	Attribute #	USINT	Position Controller Attribute	Position Controller class attribute number. Must be a settable attribute.
6	Required for Command 02	Set	Compare Time-Out value	DINT	Compare Time-out value in milliseconds.	Set from 0 to 7FFFFFFF hex. If compare does not happen within time-out a fault is generated and motion stops. 0 = no time-out.
7	Required for Command 02	Set	Compare Data	Dependent on attribute #	Compare data for end of command.	If the attribute listed above is Becomes equal to the compare data the block is done and the next link block is executed.

### 5-27.3.3 Conditional Link Greater Than Command - 03

This command is used for conditional linking or branching in a linked chain of commands. Attribute 1 must be set to 03.

**Table 5-27.5 Command Block – Conditional Link Greater Than Command**

Attr ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
3	Required for Command 03	Set	Target Class	USINT	Target class number to perform block sequencing on.	This value Defines the class which will be sequenced.
4	Required for Command 03	Set	Target Instance	USINT	Target instance number of the class to perform block sequencing on.	This value Defines the instance of the class which will be sequenced.
5	Required for Command 03	Set	Attribute #	USINT	Position Controller class attribute number. Must be a settable attribute.	
6	Required for Command 03	Set	Compare Link #	USINT	Conditional Link Number.	Alternate Link block if attribute is greater than compare data.
7	Required for Command 03	Set	Compare Data	Dependent on attribute #	Compare data for conditional link.	If the attribute listed above is greater than the compare data the normal link attribute (02) is ignored and the next block executed is the compare link block.

**5-27.3.4 Conditional Link Less Than Command - 04**

This command is used for conditional linking or branching in a linked chain of commands. Attribute 1 must be set to 04.

**Table 5-27.6 Command Block – Conditional Link Less Than Command**

Attr ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
3	Required for Command 04	Set	Target Class	USINT	Target class number to perform block sequencing on.	This value Defines the class which will be sequenced.
4	Required for Command 04	Set	Target Instance	USINT	Target instance number of the class to perform block sequencing on.	This value Defines the instance of the class which will be sequenced.
5	Required for Command 04	Set	Attribute #	USINT	Position Controller Attribute	Position Controller class attribute number. Must be a settable attribute.
6	Required for Command 04	Set	Compare Link #	USINT	Conditional Link Number.	Alternate Link block if attribute is less than compare data.
7	Required for Command 04	Set	Compare Data	Dependent on attribute #	Compare data for conditional link.	If the attribute listed above is less than the compare data the normal link attribute (02) is ignored and the next block executed is the compare link block.

**5-27.3.5 Decrement Counter Command - 05**

This command is used to decrement the Block Sequencers counter attribute used for looping. Attribute 1 must be set to 05.

**Table 5-27.7 Command Block – Decrement Counter Command**

Attr ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
No additional attributes required for this command						

**5-27.3.6 Delay Command - 06**

This command is used to perform a delay in a linked chain of commands. Attribute 1 must be set to 06.

**Table 5-27.8 Command Block – Delay Command**

Attr ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
3	Required for Command 06	Set	Delay	DINT	Delay in Milliseconds	Set the delay in milliseconds 1 hex to 7FFFFFFF hex

**5-27.3.7 Trajectory Command - 07**

This command is used to initiate a move. Attribute 1 must be set to 07.

**Table 5-27.9 Command Block – Trajectory Command**

Attr ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
3	Required for Command 07	Set	Target Position	DINT	Target Position.	Profile destination defined in position Units
4	Required for Command 07	Set	Target Velocity	DINT	Target Velocity.	Profile velocity defined in profile units per second.
5	Required for Command 07	Set	Incremental	BOOL	Absolute / Incremental flag.	0 = absolute position, 1 = incremental position

**5-27.3.8 Trajectory Command and Wait - 08**

This command is used to initiate a move and wait for completion. Attribute 1 must be set to 08.

**Table 5-27.10 Command Block – Trajectory Command and Wait**

Attr ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
3	Required for Command 08	Set	Target Position	DINT	Target Position.	Profile destination defined in position Units
4	Required for Command 08	Set	Target Velocity	DINT	Target Velocity.	Profile velocity defined in profile units per second.
5	Required for Command 08	Set	Incremental	BOOL	Absolute / Incremental flag.	0 = absolute position, 1 = relative position

**5-27.3.9 Velocity Change Command - 09**

This command is used to initiate a move and wait for completion. Attribute 1 must be set to 09.

**Table 5-27.11 Command Block – Change Command**

Attr ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
3	Required for Command 09	Set	Target Velocity	DINT	Target Velocity.	Profile velocity defined in profile units per second.

**5-27.3.10 Goto Home Command - 10**

This command is used perform a move to the captured Home position. Attribute 1 must be set to 10.

**Table 5-27.12 Command Block – Goto Home Command**

Attr ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
3	Required for Command 10	Set	Home Offset	DINT	Home Position Offset	The offset plus the captured Home position equals the absolute target position.
4	Required for Command 10	Set	Velocity	DINT	Target Velocity.	Profile velocity defined in profile units per second.

### 5-27.3.11 Goto Index Command - 11

This command is used perform a move to the captured Index position. Attribute 1 must be set to 11.

**Table 5-27.13 Command Block – Goto Index Command**

Attr ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
3	Required for Command 11	Set	Index Offset	DINT	Index Position Offset	The offset plus the captured Index position equals the absolute target position.
4	Required for Command 11	Set	Velocity	DINT	Target Velocity.	Profile velocity defined in profile units per second.

### 5-27.3.12 Goto Registration Position Command - 12

This command is used perform a move to the captured Registration position. Attribute 1 must be set to 12.

**Table 5-27.14 Command Block – Goto Registration Position Command**

Attr ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
3	Required for Command 12	Set	Registration Offset	DINT	Registration Position Offset	The offset plus the captured Registration position equals the absolute target position.
4	Required for Command 12	Set	Velocity	DINT	Target Velocity.	Profile velocity defined in profile units per second.

### 5-27.3.13 Common Services

**Table 5-27.15 Command Block Common Services**

Need in Implementation	Name	Service Code	Description of Service
Required	Get_Attribute_Single	0Ehex	Returns the contents of the specified attribute
Required	Set_Attribute_Single	10hex	Modifies the attribute value.

See Appendix A for definitions of these services

## 5-28 Motor Data Object

**Class Code: 28 hex**

This object serves as a database for motor parameters.

### 5-28.1 Class Attributes

**Table 5-28.1 Motor Data Class Attributes**

Attribute ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1 thru 7	These class attributes are optional and are described in Chapter 4 of this specification.					

### 5-28.2 Instance Attributes

The Motor Data Instance Attribute set varies depending on the type of motor that the object is representing. Instance attribute 3 “MotorType” is required, and its value determines the motor specific attributes that are available for that motor type. For all motor types, Attributes 1-5 are the same.

**Table 5-28.2 Motor Data Instance Attributes**

Attr ID	Need in Implem	Access Rule	NV	Name	Data Type	Description of Attribute
1	Conditional	Get	NV	NumAttr	USINT	Number of Attributes supported
2	Optional	Get	NV	Attributes	Array of USINT	List of attributes supported
3	Required	Set/Get	NV	MotorType	USINT	0 - Non-standard motor 1 - PM DC Motor 2 - FC DC Motor 3 - PM Synchronous Motor 4 - FC Synchronous Motor 5 - Switched Reluctance Motor 6 - Wound Rotor Induction Motor 7 - Squirrel Cage Induction Motor 8 - Stepper Motor 9 - Sinusoidal PM BL Motor 10 - Trapezoidal PM BL Motor
4	Optional	Set/Get	NV	CatNumber	SHORT_STRING	Manufacturer's Motor Catalog Number (Nameplate number) 32 chars max
5	Optional	Set/Get	NV	Manufacturer	SHORT_STRING	Manufacturer's Name 32 chars max

### 5-28.2.1 Motor Type Specific Motor Data Instance Attributes

Different motor types require different data to describe the motor. For example, AC Induction motors do not need field current data like a DC motor to describe the motor. For this reason, motor data attributes that are numbered greater than 5 are described separately for different classes of motors. The following table shows the classes of motors described in this specification. Other motor classes and types will be described in future revisions of this specification.

**Table 5-28.3 Motor Type Specific Motor Data Instance Attributes**

Motor Class	Motor Types in Class	Section Reference
AC Motor	3 - PM Synchronous 6 - Wound Rotor Induction 7 - Squirrel Cage Induction Motor	5-28.2.1.1
DC Motor	1 - PM DC Motor 2 - FC DC Motor	5-28.2.1.2

#### 5-28.2.1.1 AC Motor Instance Attributes

**Table 5-28.4 AC Motor Instance Attributes**

Attr ID	Need in Implem	Access Rule	NV	Name	Data Type	Description of Attribute
6	Required	Set/Get	NV	RatedCurrent	UINT	Rated Stator Current Units: [100mA]
7	Required	Set/Get	NV	RatedVoltage	UINT	Rated Base Voltage Units: [V]
8	Optional	Set/Get	NV	RatedPower	UDINT	Rated Power at Rated Freq Units: [W]
9	Optional	Set/Get	NV	RatedFreq	UINT	Rated Electrical Frequency Units: [Hz]
10	Optional	Set/Get	NV	RatedTemp	UINT	Rated Winding Temperature Units: [ degrees C]
11	Optional	Set/Get	NV	MaxSpeed	UINT	Maximum allowed motor speed Units: [RPM]
12	Optional	Set/Get	NV	PoleCount	UINT	Number of poles in the motor.
13	Optional	Set/Get	NV	TorqConstant	UDINT	Motor torque constant Units: [0.001 x Nm/A]
14	Optional	Set/Get	NV	Inertia	UDINT	Rotor Inertia Units: [ $10^{-6}$ x kg.m <sup>2</sup> ]
15	Optional	Set/Get	NV	BaseSpeed	UINT	Nominal speed at rated frequency from nameplate Units: [RPM]
19	Optional	Set/Get	NV	ServiceFactor	USINT	Units: [%] Range: 0 .. 255
20	Optional	Get	NV	International CatNumber	STRINGI	Catalog Name or Number
21	Optional	Get	NV	International Manufacturer Name	STRINGI	Name of Motor Manufacturer
22	Optional	Set	NV	SerialNumber	SHORT STRING	Name Plate Serial Number of Motor (32 characters or less)
23	Optional	Set	NV	TagName	SHORT STRING	Plant's Tag Name of Motor (32 characters or less)
24	Optional	Set	NV	SupplyStyle	BOOL	0 = three phase, 1 = single phase
25	Optional	Set	NV	TimeRating	UINT	Maximum Continuous Operation Time (in minutes)

**Motor Data Object, Class Code: 28<sub>Hex</sub>**

<b>Attr ID</b>	<b>Need in Implem</b>	<b>Access Rule</b>	<b>NV</b>	<b>Name</b>	<b>Data Type</b>	<b>Description of Attribute</b>
26	Optional	Set	NV	InrushCurrent	UINT	Maximum inrush current, Units: [100mA]
27	Optional	Set	NV	LockedRotor CodeLetter	SHORT STRING	NEMA MG-1 Code Letter [MG1-10.37.2] The single ASCII character corresponding to the Code Letter for Locked Rotor kVA
28	Optional	Set	NV	DesignLetter	SHORT STRING	IEC or NEMA Design Letters
29	Optional	Set	NV	Thermal Protection	BOOL	TRUE when motor is marked "Thermally Protected"

**5-28.2.1.2 DC Motor Instance Attributes****Table 5-28.5 DC Motor Instance Attributes**

<b>Attr ID</b>	<b>Need in Implem</b>	<b>Access Rule</b>	<b>NV</b>	<b>Name</b>	<b>Data Type</b>	<b>Description of Attribute</b>
6	Required	Set/Get	NV	RatedCurrent	UINT	Rated Armature Current Units: [100mA]
7	Required	Set/Get	NV	RatedVoltage	UINT	Rated Armature Voltage Units: [V]
8	Optional	Set/Get	NV	RatedPower	UDINT	Rated Power at MaxSpeed Units: [W]
10	Optional	Set/Get	NV	RatedTemp	UINT	Rated Winding Temperature Units: [ degrees C]
11	Optional	Set/Get	NV	MaxSpeed	UINT	Maximum allowed motor speed Units: [RPM]
13	Optional	Set/Get	NV	TorqConstant	UDINT	Motor torque constant Units: [0.001 x Nm/A]
14	Optional	Set/Get	NV	Inertia	UDINT	Rotor Inertia Units: [10 <sup>-6</sup> x kg.m <sup>2</sup> ]
15	Optional	Set/Get	NV	BaseSpeed	UINT	Nominal speed at rated voltage Units: [RPM]
16	Optional	Set/Get	NV	RatedFieldCur	UDINT	Rated Field Current Units: [mA]
17	Optional	Set/Get	NV	MinFieldCur	UDINT	Minimum Field Current Units: [mA]
18	Optional	Set/Get	NV	RatedFieldVolt	UINT	Rated Field Voltage Units: [V]
20	Optional	Get	NV	International CatNumber	STRINGI	Catalog Name or Number
21	Optional	Get	NV	International Manufacturer Name	STRINGI	Name of Motor Manufacturer
22	Optional	Set	NV	SerialNumber	SHORT STRING	Name Plate Serial Number of Motor (32 characters or less)

**Motor Data Object, Class Code: 28<sub>Hex</sub>**

<b>Attr ID</b>	<b>Need in Implem</b>	<b>Access Rule</b>	<b>NV</b>	<b>Name</b>	<b>Data Type</b>	<b>Description of Attribute</b>
23	Optional	Set	NV	TagName	SHORT STRING	Plant's Tag Name of Motor (32 characters or less)
24	Optional	Set	NV	SupplyStyle	BOOL	0 = three phase, 1 = single phase
25	Optional	Set	NV	TimeRating	UINT	Maximum Continuous Operation Time (in minutes)
26	Optional	Set	NV	InrushCurrent	UINT	Maximum inrush current, Units: [100mA]
27	Optional	Set	NV	LockedRotor CodeLetter	SHORT STRING	NEMA MG-1 Code Letter [MG1-10.37.2] The single ASCII character corresponding to the Code Letter for Locked Rotor kVA
28	Optional	Set	NV	DesignLetter	SHORT STRING	IEC or NEMA Design Letters
29	Optional	Set	NV	Thermal Protection	BOOL	TRUE when motor is marked "Thermally Protected"

The following engineering abbreviations are used in the above Motor Data Instance Attribute descriptions.

**Table 5-28.6 Abbreviations Used in This Object**

<b>Abbreviation</b>	<b>Description</b>
mA	milli Amps
V	Volts
RPM	Revolutions Per Minute
Kg.m <sup>2</sup>	kilograms times meters squared
Nm/A	Newton meters per Amp
Degrees C	degrees Centigrade
Hz	Hertz
W	Watts

**5-28.3 Common Services****Table 5-28.7 Motor Data Object Common Services**

<b>Service Code</b>	<b>Need in Implementation</b>		<b>Service Name</b>	<b>Description of Service</b>
	<b>Class</b>	<b>Instance</b>		
0E <sub>hex</sub>	Conditional*	Required	Get_Attributes_Single	Returns the contents of the specified attribute.
10 <sub>hex</sub>	n/a	Required	Set_Attributes_Single	Modifies an attribute value.
15 <sub>hex</sub>	n/a	Optional	Restore	Restores attribute values from a storage location accessible by the Save service. Service is only available if the Control Supervisor Object of the drive is in the <b>Ready</b> or <b>Wait_Power</b> states.
16 <sub>hex</sub>	n/a	Optional	Save	Saves all attribute values to a location accessible by the Restore service.

\* The Get\_Attribute\_Single service is **required** at the class level if any class attributes are implemented

See Appendix A for definitions of these common services.

#### **5-28.4 Object-specific Services**

The Motor Data object provides no object specific services.

#### **5-28.5 Behavior**

The Motor Data Object serves only as an internal database for motor parameters, which are accessed by other objects in the drive.

This page is intentionally left blank

## 5-29 Control Supervisor Object

**Class Code: 29 hex**

This object models all the management functions for devices within the “Hierarchy of Motor Control Devices”. The behavior of motor control devices is described by the State Transition Diagram and the State Event Matrix (see Section 5-29.5).

### 5-29.1 Class Attributes

**Table 5-29.1 Control Supervisor Object Class Attributes**

Attribute ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1 thru 7	These class attributes are optional and are described in Chapter 4 of this specification.					

### 5-29.2 Instance Attributes

**Table 5-29.2 Control Supervisor Object Instance Attributes**

Attr ID	Need in Implem	Access Rule	NV	Name	Data Type	Description of Attribute
1	Optional	Get		NumAttr	USINT	Number of Attributes supported
2	Optional	Get		Attributes	Array of USINT	List of attributes supported
3	Required	Set		Run1	BOOL	See Run/Stop Event Matrix
4	Optional	Set		Run2	BOOL	See Run/Stop Event Matrix
5	Optional	Set		NetCtrl	BOOL	Requests Run/Stop control to be local or from network. 0 = Local Control 1 = Network Control Note that the actual status of Run/Stop control is reflected in attribute 15, CtrlFromNet.
6	Optional	Get		State	USINT	0 = Vendor Specific 1 = Startup 2 = Not_Ready 3 = Ready 4 = Enabled 5 = Stopping 6 = Fault_Stop 7 = Faulted
7	Required for Drives and Servos only	Get		Running1	BOOL	1 = (Enabled <b>and</b> Run1) <b>or</b> (Stopping <b>and</b> Running1) <b>or</b> (Fault_Stop <b>and</b> Running1) 0 = Other state

Control Supervisor Object, Class Code: 29<sub>Hex</sub>

Attr ID	Need in Implem	Access Rule	NV	Name	Data Type	Description of Attribute
8	Optional	Get		Running2	BOOL	1 = (Enabled <b>and</b> Run2) <b>or</b> (Stopping <b>and</b> Running2) <b>or</b> (Fault_Stop <b>and</b> Running2) 0 = Other state
9	Optional	Get		Ready	BOOL	1 = Ready <b>or</b> Enabled <b>or</b> Stopping 0 = Other state
10	Required for Drives and Servos only	Get		Faulted	BOOL	1 = Fault Occurred (latched) 0 = No Faults present
11	Optional	Get		Warning	BOOL	1 = Warning (not latched) 0 = No Warnings present If warnings are not supported, this attribute should always be 0
12	Required for Drives and Servos only	Set		FaultRst	BOOL	0->1 = Fault Reset 0 = No action
13	Optional	Get		FaultCode	UINT	If in <b>Faulted</b> state, FaultCode indicates the fault that caused the transition to <b>Faulted</b> state. If multiple faults occurred simultaneously, the vendor chooses which to report, and the rest are lost. If not in <b>Faulted</b> state, FaultCode indicates the fault that caused the last transition to the <b>Faulted</b> state.  Power up state of fault code is vendor specific.  Fault codes for drives are different than fault codes for starters.  See appropriate device profile for fault codes.
14	Optional	Get		WarnCode	UINT	Code word indicating warning present. If multiple warnings are present, the lowest code value is displayed.  Note that fault codes for drives and servos are different than fault codes for starters.  See appropriate device profile for fault codes.
15	Optional	Get		CtrlFromNet	BOOL	Status of Run/Stop control source. 0=Control is local 1=Control is from network
16	Optional	Set	*	NetFaultMode	USINT	Action on loss of CIP Network 0 = Fault + Stop 1 = Ignore (Warning Optional) 2= Vendor specific
17	Optional	Set		ForceFault/Trip	BOOL	0 ->1 = Force
18	Optional	Get		ForceStatus	BOOL	0 = Not Forced Nonzero = Forced

Control Supervisor Object, Class Code: 29<sub>Hex</sub>

Attr ID	Need in Implem	Access Rule	NV	Name	Data Type	Description of Attribute
19	Optional	Set		Delay *	ITIME	Time delay before any fault or idle mode action occurs after the fault or idle event is detected.
20	Optional	Set		NetIdleMode *	USINT	Mode on reception of CIP communication IDLE event – See semantics below.
21	Optional	Set		ProtectMode *	USINT	Mode on detection of a motor protection event – See semantics below.
22	Optional	Set		CycleCount *	UDINT	Number of operations (motor starts) on the equipment. This value shall be zero at the time of manufacture. This value may be settable to other values. This attribute is incremented each time a motor start activity occurs (eg: the starter causes a motor to begin turning, change direction or speed)
23	Conditional	Set	NV	Fault/Warning Code Style	USINT	Identifies the error and warning codes produced by this instance of Control Supervisor Default = 0

\* these attribute values shall be maintained through power cycles

### 5-29.2.1 Operation Mode Values

The three modes of operation attributes above (number 16, 20 and 21) describe the mode or action that the control supervisor shall assume after specific events. These modes of operation are also coupled with the Delay attribute. The attribute semantics of the NetFaultMode, NetIdleMode and ProtectionMode are as enumerated below:

Table 5-29.3 Operation Mode Values

Mode	Action	Error/Warning
0	Stop	Fault Code
1	Ignore (Warning Optional)	Warning Code optional
2	Vendor specific action	Error and Warning optional
3	Run 1 (Invalid for ProtectMode Attribute)	Fault Code
4	Run 2 (Invalid for ProtectMode Attribute)	Fault Code
5 thru 99	Reserved by CIP	
100 thru 199	Vendor specific action	Error and Warning optional
200 thru 255	Reserved by CIP	

### 5-29.2.2 NetFaultMode Attribute

The NetFaultMode attribute establishes the mode of operation (see Table 5-29.3, Operation Mode Values) on loss of network communication. The default value for this attribute is zero. If the attribute is not implemented, the behavior of the control supervisor shall be the “Stop” mode.

If the Delay attribute is both implement and non-zero, the mode specified will be active only after a time specified by the delay has expired.

### 5-29.2.3 NetIdleMode Attribute

The NetIdleMode attribute establishes the mode of operation (see Table 5-29.3, Operation Mode Values) on reception of network Idle communication. The default value for this attribute is zero. If this attribute is not implemented, the behavior of the control supervisor shall be the “Stop” mode.

If the Delay attribute is both implement and non-zero, the mode specified will be active only after a time specified by the delay has expired.

### 5-29.2.4 ProtectMode Attribute

The ProtectMode attribute establishes the mode of operation (see Table 5-29.3, Operation Mode Values) on reception of a motor protection event. The default value for this attribute is zero. If this attribute is not implemented, the behavior of the control supervisor shall be the “Stop” mode.

If the Delay attribute is both implement and non-zero, the mode specified will be active only after a time specified by the delay has expired.

### 5-29.2.5 Fault/Warning Code Style Attribute

The Fault/Warning Code Style attribute is used to indicate the identification of which types of Fault and/or Warning codes are produced by this instance of the Control Supervisor. This attribute is optional except in instances where the Fault/Warning Codes utilized is not defined by the device profile or is different than the enumerations expected due to the reported device profile.

This attribute has the values shown in Table 5-29.4.

**Table 5-29.4 Fault/Warning Code Styles**

Value	Meaning	Reference
0	Default value. The Fault and Warning Codes presented are those implied by the device profile.	Table 5-29.5
1	The DRIVECOM (16-bit) codes are utilized	Table 5-29.8
2	The Abbreviated (8-bit) codes are utilized	Table 5-29.9
2 thru 255	Reserved by CIP	

The implied Fault and Warning Code usage broken down by device profile are shown in Table 5-29.5.

**Table 5-29.5 Implied Fault/Warning Code Usage by Device Profile**

Value	Meaning	Reference
02 <sub>hex</sub>	AC Drive	DRIVECOM (16-bit) code set
not assigned	Circuit Breaker	Abbreviated (8-bit) code set
not assigned	Circuit Disconnect	Abbreviated (8-bit) code set
15 <sub>hex</sub>	Contactor	Abbreviated (8-bit) code set
13 <sub>hex</sub>	DC Drive	DRIVECOM (16-bit) code set
not assigned	Fuse	Abbreviated (8-bit) code set

**Control Supervisor Object, Class Code: 29<sub>Hex</sub>**

<b>Value</b>	<b>Meaning</b>	<b>Reference</b>
03 <sub>hex</sub>	Motor Overload	Abbreviated (8-bit) code set
16 <sub>hex</sub>	Motor Starter	Abbreviated (8-bit) code set
not assigned	Power Monitor	Abbreviated (8-bit) code set
not assigned	Servo Drive	DRIVECOM (16-bit) code set
17 <sub>hex</sub>	Soft Starter	Abbreviated (8-bit) code set

**5-29.3 Common Services****Table 5-29.6 Control Supervisor Object Common Services**

<b>Service Code</b>	<b>Need in Implementation</b>		<b>Service Name</b>	<b>Description of Service</b>
	<b>Class</b>	<b>Instance</b>		
0E <sub>hex</sub>	Conditional	Required	Get_Attributes_Single	Returns the contents of the specified attribute.
10 <sub>hex</sub>	n/a	Required	Set_Attributes_Single	Modifies an attribute value.
05 <sub>hex</sub>	n/a	Required	Reset	Resets the drive to the <b>start-up</b> state.

See Appendix A for description of these services

**5-29.4 Object Specific Services**

The Control Supervisor object provides no object specific services.

**5-29.5 Behavior**

The State Transition Diagram provides a graphical description of the states and corresponding state transitions. The State Event Matrix lists all events and the corresponding action to be taken while in each state. A subset of states and events may be supported in an application but the behavior must be consistent.

Figure 5-29.1 Control Supervisor Object State Transition Diagram

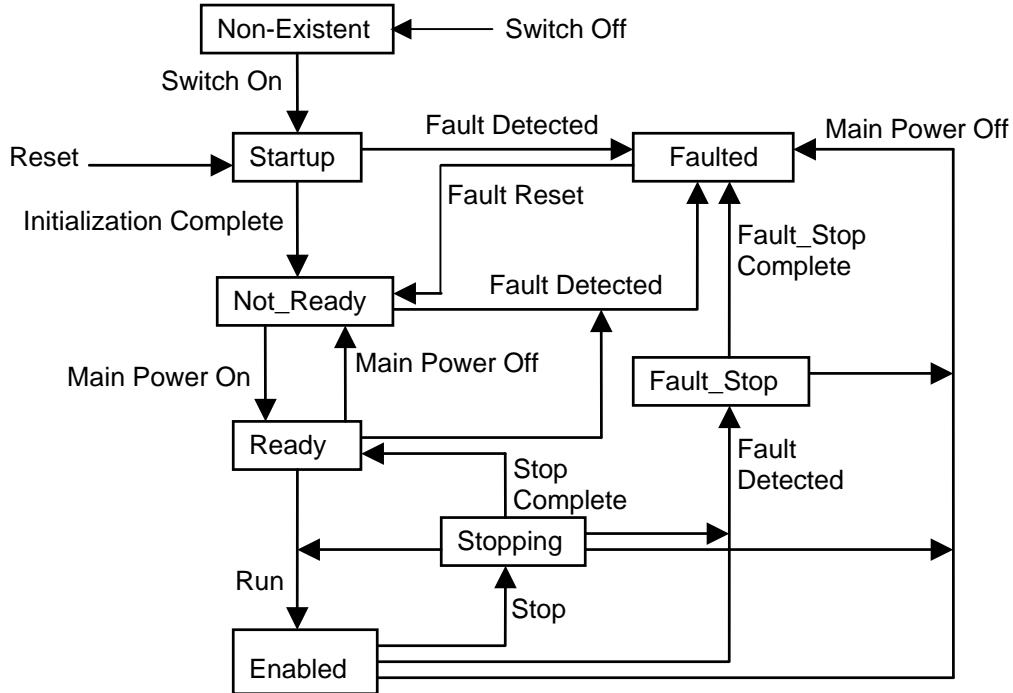


Table 5-29.7 Control Supervisor Object State Event Matrix

Event	State								
	Non_Exist	Startup	Not_Ready	Ready	Enabled	Stopping	Fault_Stop	Faulted	
Switch Off	N/A	Transition to Non_Exist	Transition to Non_Exist	Transition to Non_Exist	Transition to Non_Exist	Transition to Non_Exist	Transition to Non_Exist	Transition to Non_Exist	
Switch on	Transition to Startup	N/A	N/A	N/A	N/A	N/A	N/A	N/A	
Initialization Complete	N/A	Transition to Not_Ready	N/A	N/A	N/A	N/A	N/A	N/A	
Main Power On	N/A	N/A	Transition to Ready	N/A	N/A	N/A	N/A	N/A	
Run *	N/A	N/A	N/A	Transition to Enabled	N/A	Transition to Enabled	N/A	N/A	
Stop *	N/A	N/A	N/A	N/A	Transition to Stopping	N/A	N/A	N/A	
Stop Complete	N/A	N/A	N/A	N/A	N/A	Transition to Ready	N/A	N/A	
Reset	N/A	N/A	Transition to Startup	Transition to Startup	Transition to Startup	Transition to Startup	Transition to Startup	Transition to Startup	
Main Power Off	N/A	N/A	N/A	Transition to Not_Ready	Transition to Faulted	Transition to Faulted	Transition to Faulted	N/A	
Fault Detected	N/A	Transition to Faulted	Transition to Faulted	Transition to Faulted	Transition to Fault_Stop	Transition to Fault_Stop	N/A	N/A	
Fault_Stop Complete	N/A	N/A	N/A	N/A	N/A	N/A	Transition to Faulted	N/A	
Fault Reset	N/A	N/A	N/A	N/A	N/A	N/A	N/A	Transition to Not_Ready	

\* See section 5-29.5.1 for further explanation of these events and how they are generated.

### 5-29.5.1 Run/Stop Event Matrix

Attribute 5, NetCtrl is used to request that Run Stop events be controlled from the network. The device however, has the option of inhibiting Run Stop events from the network, as the user/application may not allow Run Stop control from the network under certain circumstances. Only when attribute 15, CtrlFromNet is set to 1 by the device in response to a NetCtrl request, is Run Stop control actually accomplished from the network.

If attribute 15, CtrlFromNet is 1, the events Run and Stop are triggered by a combination of the Run1 and Run2 attributes as shown in the following table. Note that Run1 and Run2 have different contexts for different device types. The following table shows the Run1 and Run2 contexts for the devices within the motor control hierarchy.

**Table 5-29.8 Run1/Run2 Contexts**

	Starters					Drives and Servos
	Contactor	Starter	Reverser	2 Speed	SoftStart	
Run1	Close	Run	RunFwd	RunLo	RunRamp1	RunFwd
Run2	NA	NA	RunRev	RunHigh	RunRamp2	RunRev

If CtrlFromNet is 0, Run and Stop events must be controlled using local input(s) provided by the vendor. The Run and Stop events effect drive behavior as shown in the State Transition Diagram and the State Event Matrix.

**Table 5-29.9 Run1 and Run2 Triggers**

Run1A	Run2	Trigger Event	Run Type
0	0	Stop	NA
0 -> 1	0	Run	Run1
0	0 -> 1	Run	Run2
0 -> 1	0 -> 1	No Action	NA
1	1	No Action	NA
1->0	1	Run	Run2
1	1->0	Run	Run1

**Important note:** Local stop and run signals could override or be interlocked with the run/stop control through CIP. These are vendor specific features. Vendors should explain these interlocks and overrides in their product documentation.

### 5-29.6 Fault and Warning codes

This table lists the fault and warning codes used by AC Drives, DC Drives, and Servos for the ‘Control Supervisor’ object. The source of the fault codes is the DRIVECOM Nutzergruppe e.V, which has granted permission to ODVA to use the fault codes in ODVA device profiles.

**Table 5-29.10 DRIVECOM (16-bit) Fault and Warning Codes**

Code Value [Hex]	Meaning
0000	No fault
1000	General Fault
2000	Current
2100	Current, Device Input Side
2110	Short Circuit/Short to Earth

Control Supervisor Object, Class Code: 29<sub>Hex</sub>

Code Value [Hex]	Meaning
2120	Short to Earth
2121	Short to earth in Phase L1
2122	Short to earth in Phase L2
2123	Short to earth in Phase L3
2130	Short Circuit
2131	Short Circuit in Phases L1 -L2
2132	Short Circuit in Phases L2-L3
2133	Short Circuit in Phases L3-L1
2200	Current Inside the Device
2211	Current inside the device, No. 1
2212	Current inside the device, No. 2
2213	Overcurrent during Startup
2214	Overcurrent during Slowdown
2220	Continuous Overcurrent
2221	Continuous Overcurrent No. 1
2222	Continuous Overcurrent No. 2
2230	Short Circuit/Short to earth
2240	Short to earth
2250	Short Circuit
2300	Current, Device Output Side
2310	Continuous Overcurrent
2311	Continuous Overcurrent, No. 1
2312	Continuous Overcurrent, No. 2
2320	Short Circuit/Short to Earth
2330	Short to Earth
2331	Short to Earth in Phase U
2332	Short to Earth in Phase V
2333	Short to Earth in Phase W
2340	Short Circuit
2341	Short Circuit in Phases U-V
2342	Short Circuit in Phases V-W
2343	Short Circuit in Phases W-U
<b>3000</b>	<b>Voltage</b>
3100	Mains Voltage
3110	Mains overvoltage
3111	Mains overvoltage in phase L1
3112	Mains overvoltage in phase L2
3113	Mains overvoltage in phase L3
3120	Mains undervoltage
3121	Mains undervoltage in phase L1
3122	Mains undervoltage in phase L2
3123	Mains undervoltage in phase L3
3130	Phase Failure
3131	Failure of Phase L1
3132	Failure of Phase L2
3133	Failure of Phase L3
3134	Phase Sequence
3140	Mains Frequency
3141	Mains Frequency too high
3142	Mains Frequency too low
3200	Voltage inside the Device
3210	Overvoltage inside the device
3211	Overvoltage No. 1
3212	Overvoltage No. 2

Control Supervisor Object, Class Code: 29<sub>Hex</sub>

Code Value [Hex]	Meaning
3220	Undervoltage inside the Device
3221	Undervoltage No. 1
3222	Undervoltage No. 2
3230	Charging Error
3300	Output Voltage
3310	Output Ovvoltage
3311	Output ovvoltage in Phase U
3312	Output ovvoltage in Phase V
3313	Output ovvoltage in Phase W
3320	Armature Circuit
3321	Armature Circuit Discontinuity
3330	Field Circuit
3331	Field Circuit Discontinuity
<b>4000</b>	<b>Temperature</b>
4100	Ambient Temperature
4110	Excess Ambient Temperature
4120	Inadequate Ambient Temperature
4130	Ingoing Ambient Temperature
4140	Outgoing Air Temperature
4200	Device Temperature
4210	Excess Device Temperature
4220	Inadequate Device Temperature
4300	Drive Temperature
4310	Excess Drive Temperature
4320	Inadequate Drive Temperature
4400	Power Supply Temperature
4410	Excess Supply Temperature
4420	Inadequate Supply Temperature
<b>5000</b>	<b>Device Hardware</b>
5100	Power Supply
5110	Low Voltage Power Supply
5111	± 15V Power Supply
5112	+24V Power Supply
5113	+5V Power Supply
5120	DC Link Power Supply
5200	Control
5210	Measurement Circuit
5220	Computing Circuit
5300	Operator Control Circuit
5400	Power Section
5410	Output Stages
5420	Chopper
5430	Input Stages
<b>6000</b>	<b>Device Software</b>
6010	Software Reset (Watchdog)
6100	Internal Software
6200	User Software
6300	Date Set
6301	Data Set No.1
6302 ... 630E	from 2 to 14 accordingly
630F	Data Set No.15
6310	Parameter Loss

Control Supervisor Object, Class Code: 29<sub>Hex</sub>

Code Value [Hex]	Meaning
6320	Parameter Error
<b>7000</b>	<b>Additional Modules</b>
7100	Power
7110	Brake Chopper
7111	Brake Chopper Failure
7112	Brake Chopper overcurrent
7113	Brake Chopper Wiring
7120	Motor
7121	Motor Blocked
7200	Measurement Circuit
7300	Sensor
7301	Tacho defective
7302	Wrong Tacho Polarity
7303	Resolver 1 defective
7304	Resolver 2 defective
7305	Incremental Encoder 1 Defective
7306	Incremental Encoder 2 Defective
7307	Incremental Encoder 3 Defective
7310	Speed
7320	Position
7400	Computing Circuit
7410	Velocity Ramp Generation
7420	Trajectory Generation Error
7421	Invalid Parameters
7422	Trajectory Generator Precalculation
7423	Trajectory Interpolation
7500	Communication
7510	Serial Interface No 1
7520	Serial Interface No 2
7600	Data Memory
<b>8000</b>	<b>Monitoring</b>
8100	Communication
8110	Operational Data Monitoring
8111	No SYNC
8112	Synchronisation Fault
8113	No COMMAND
8114	COMMAND outside window.
8115	Cannot transmit ACTUAL
8116	ACTUAL outside window.
8120	Host Monitoring
8200	Closed Loop Control
8300	Torque Controller
8302	Torque Limiting
8311	Excess Torque
8312	Heavy Starting
8313	Standstill Torque
8321	Inadequate Torque
8331	Torque Breakage
8400	Speed Controller
8401	Velocity Following Error
8402	Velocity Limiting
8500	Position Controller
8501	Position Following Error

Control Supervisor Object, Class Code: 29<sub>Hex</sub>

Code Value [Hex]	Meaning
8502	Position Limiting
8600	Positioning Controller
8611	Following Error
8612	Reference Limit
8700	Synchro Controller
8800	Winding Controller
<b>9000</b>	<b>External Malfunction</b>
<b>F000</b>	<b>Additional Functions</b>
F001	Deceleration
F002	Inadequate Synchronization
F003	Lifting Mechanism
F004	Open LOOP Control

Table 5-29.11 Abbreviated (8-bit) Fault and Warning Codes

Code Value	Meaning
0	NO FAULT
10	TEST
11	FORCE TRIP
12	E-STOP
13	RESTART RETRIES
14	SHORT CIRCUIT
15	AUXILLIARY I/O FAULT
16	NETWORK POWER
20	CURRENT TRIP
21	THERMAL OVERLOAD
22	PHASE LOSS
23	PHASE LOSS - A
24	PHASE LOSS - B
25	PHASE LOSS - C
26	PHASE IMBALANCE
27	GROUND FAULT
28	JAM
29	UNDERLOAD
30	MOTOR PLUG ATTEMPT
31	STALL
32	PHASE TO PHASE SHORT
33	PHASE TO GROUND SHORT
40	CONTROL VOLTAGE
41	CONTROL UNDERTHOLD
42	CONTROL OVERVOLTAGE
43	FREQUENCY
44	SPEED SWITCH
50	MAIN VOLTAGE
51	UNDERTHOLD
52	OVERVOLTAGE
53	IMBALANCE
54	PHASE REVERSAL
55	FREQUENCY
56	OVERPOWER

Control Supervisor Object, Class Code: 29<sub>Hex</sub>

Code Value	Meaning
57	UNDERPOWER
58	VAR
60	HARDWARE
61	ILLEGAL FLC SETTING
62	MEMORY FAULT
63	HARDWARE LINK FAULT
64	NO DEVICE POWER
65	WATCHDOG TIMEOUT
66	INTERNAL COMPONENT OVER TEMPERATURE
67	TEMPERATURE SENSOR FAULT
68	TEMPERATURE SENSOR LOSS
69	INTERNAL COMMUNICATION/CONTROL LOST
70	MISC
71	FAIL TO CLOSE
72	FAIL TO OPEN
73	STARTS/HOUR EXCEEDED
74	LOW POWER FACTOR
75	STATOR OVERTEMP
76	BEARING OVERTEMP
77	INCOMPLETE SEQUENCE
78	CONTACTOR #2, FAILED TO CLOSE
79	CONTACTOR #2, FAILED TO OPEN
80	CONTACTOR #3, FAILED TO CLOSE
81	CONTACTOR #3, FAILED TO OPEN
82	STATOR WINDING SHORT
83	MOTOR BEARING FAULT
84	EXCESSIVE VIBRATION
85	UNSTABLE DRIVEN LOAD
86	LOOSE MOTOR ROTOR BAR
87	CENTRIFUGAL UNBALANCE
88	ROTATIONAL UNBALANCE
89	MOTOR/LOAD MISALIGNMENT
90	MOTOR OPERATING OUTSIDE LOAD PROFILE
91	HIGH HARMONIC DISTORTION
92	MOTRING (MOTOR=GENERATOR OR NEGATIVE CURRENTS)

\* Note Fault Codes 101 to 255 are Vendor Specific Codes

## 5-29.7 Object-specific General and Extended Status Codes

### 5-29.12 Object-specific General and Extended Status Codes

General Status Codes (in hex)	8-bit/16-bit Associated Extended Status Codes	Status Name	Description of Status
00 - CF		General Codes	Defined by CIP specification in Appendix B
	00 - FE		Reserved Extended Status Codes
	F0 - FE		Vendor specific Extended Codes
	FF		Used with all General Codes when required and no other Extended Code is assigned
D0		DRIVECOM Warning Code	The Extended Status Code is a DRIVECOM (16-bit) Warning code
	See Table 5-29.10		DRIVECOM (16-bit) Warning Code
D1		DRIVECOM Fault Code	The Extended Status Code is a DRIVECOM (16-bit) Fault code
	See Table 5-29.10		DRIVECOM (16-bit) Fault Code
D2		Abbreviated Warning Code	The Extended Status Code is an Abbreviated (8-bit) Warning Code
	See Table 5-29.11		Abbreviated (8-bit) Warning Code
D3		Abbreviated Fault Code	The Extended Status Code is an Abbreviated (8-bit) Warning Code
	See Table 5-29.11		Abbreviated (8-bit) Warning Code

## 5-30 AC/DC Drive Object

### Class Code: 2A hex

This object models the functions specific to an AC or DC Drive. e.g. speed ramp, torque control etc.

#### 5-30.1 Class Attributes

**Table 5-30.1 AC/DC Drive Object Class Attributes**

Attribute ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1 thru 7					These class attributes are optional and are described in Chapter 4 of this specification.	

#### 5-30.2 Instance Attributes

**Table 5-30.2 AC/DC Drive Object Instance Attributes**

ttr ID	Need in Implem	Access Rule	NV	Name	Data Type	Description of Attribute
1	Optional	Get		NumAttr	USINT	Number of Attributes supported
2	Optional	Get		Attributes	Array of USINT	List of Attributes supported
3	Optional	Get		AtReference	BOOL	1 = Drive actual at reference (speed or torque reference) based on mode
4	Required	Set/ Get		NetRef	BOOL	Requests torque or speed reference to be local or from the network. 0 = Set Reference not DN Control 1 = Set Reference at DN Control Note that the actual status of torque or speed reference is reflected in attribute 29, RefFromNet.
5	Optional	Set/ Get		NetProc	BOOL	Requests process control reference to be local or from the network. 0 = Set Process not DN Control 1 = Set Process at DN Control Note that the actual status of the process control reference is reflected in attribute 30, ProcFromNet.
6	Required	Set/ Get		DriveMode	USINT	0 = Vendor specific mode 1 = Open loop speed (Frequency) 2 = Closed loop speed control 3 = Torque control 4 = Process control (e.g. PI) 5 = Position control
7	Required	Get		SpeedActual	INT	Actual drive speed (best approximation) Units: RPM / 2 <sup>SpeedScale</sup> where SpeedScale is attribute 22
8	Required	Set/ Get		SpeedRef	INT	Speed reference Units: RPM / 2 <sup>SpeedScale</sup> where SpeedScale is attribute 22

AC/DC Drive Object, Class Code: 2A<sub>Hex</sub>

ttr ID	Need in Implem	Access Rule	NV	Name	Data Type	Description of Attribute
9	Optional	Get		CurrentActual	INT	Actual motor phase current Units: 100ma / 2 <sup>CurrentScale</sup> where CurrentScale is attribute 23
10	Optional	Set/ Get		CurrentLimit	INT	Motor phase current limit Units: 100ma / 2 <sup>CurrentScale</sup> where CurrentScale is attribute 23
11	Optional	Get		TorqueActual	INT	Actual torque Units: Nm / 2 <sup>TorqueScale</sup> where TorqueScale is attribute 24
12	Optional	Set/ Get		TorqueRef	INT	Torque reference Units: Nm / 2 <sup>TorqueScale</sup> where TorqueScale is attribute 24
13	Optional	Get		ProcessActual	INT	Actual process control value Units: % / 2 <sup>ProcessScale</sup> where ProcessScale is attribute 25
14	Optional	Set/ Get		ProcessRef	INT	Process control reference set point Units: % / 2 <sup>ProcessScale</sup> where ProcessScale is attribute 25
15	Optional	Get		PowerActual	INT	Actual output power Units: Watts / 2 <sup>PowerScale</sup> where PowerScale is attribute 26
16	Optional	Get		InputVoltage	INT	Input Voltage Units: Volts / 2 <sup>VoltageScale</sup> where VoltageScale is attribute 27
17	Optional	Get		OutputVoltage	INT	Output Voltage Units: Volts / 2 <sup>VoltageScale</sup> where VoltageScale is attribute 27
18	Optional	Set/ Get		AccelTime	UINT	Acceleration time Time from 0 to HighSpdLimit Units: ms / 2 <sup>TimeScale</sup> where TimeScale is attribute 28 Acceleration time selection for negative direction is vendor specific.
19	Optional	Set/ Get		DecelTime	UINT	Deceleration time Time from 0 to HighSpdLimit Units: ms / 2 <sup>TimeScale</sup> where TimeScale is attribute 28 Deceleration time selection for negative direction is vendor specific.
20	Optional	Set/ Get		LowSpdLimit	UINT	Minimum speed limit Units: RPM / 2 <sup>SpeedScale</sup> where SpeedScale is attribute 22
21	Optional	Set/ Get		HighSpdLimit	UINT	Maximum speed limit Units: RPM / 2 <sup>SpeedScale</sup> where SpeedScale is attribute 22
22	Optional*	Set/ Get		SpeedScale	SINT	Speed scaling factor. Scaling is accomplished as follows: ScaledSpeed = RPM / 2 <sup>SpeedScale</sup> Range: -128 .. 127
23	Optional*	Set/ Get		CurrentScale	SINT	Current scaling factor. Scaling is accomplished as follows: ScaledCurrent = A / 2 <sup>CurrentScale</sup> Range: -128 .. 127

AC/DC Drive Object, Class Code: 2A<sub>Hex</sub>

ttr ID	Need in Implem	Access Rule	NV	Name	Data Type	Description of Attribute
24	Optional*	Set/ Get		TorqueScale	SINT	Torque scaling factor. Scaling is accomplished as follows: ScaledTorque = Nm / $2^{\text{TorqueScale}}$ Range: -128 .. 127
25	Optional*	Set/ Get		ProcessScale	SINT	Process scaling factor. Scaling is accomplished as follows: ScaledProcess = % / $2^{\text{ProcessScale}}$ Range: -128 .. 127
26	Optional*	Set/ Get		PowerScale	SINT	Power scaling factor. Scaling is accomplished as follows: ScaledPower = W / $2^{\text{PowerScale}}$ Range: -128 .. 127
27	Optional*	Set /Get		VoltageScale	SINT	Voltage scaling factor. Scaling is accomplished as follows: ScaledVoltage = V / $2^{\text{VoltageScale}}$ Range: -128 .. 127
28	Optional*	Set/ Get		TimeScale	SINT	Time scaling factor. Scaling is accomplished as follows: ScaledTime = ms / $2^{\text{TimeScale}}$ Range: -128 .. 127
29	Optional	Get		RefFromNet	BOOL	Status of torque/speed reference 0=Local torque/speed reference 1=Network torque/speed reference
30	Optional	Get		ProcFromNet	BOOL	Status of process control reference 0=Local process reference 1=Network process reference
31	Optional	Set/ Get		FieldIorV	BOOL	Selects Field Voltage or Field Current control for a DC Drive. 0=Voltage Control (Open Loop) 1=Current Control (Magnetizing field for DC drive)
32	Optional	Set/ Get		FieldVoltRatio	UINT	For voltage control of a DC Drive
33	Optional	Set/ Get		FieldCurSetPt	UINT	DC Drive Field Current set point. Units: Amps / $2^{\text{CurrentScale}}$ where CurrentScale is attribute 23
34	Optional	Set/ Get		FieldWkEnable	BOOL	Enables/Disables field weakening for a DC Drive 0=Disabled (DC Drive in current control) 1=Enabled
35	Optional	Get		FieldCurActual	INT	Actual Field Current for a DC Drive. Units: Amps / $2^{\text{CurrentScale}}$ where CurrentScale is attribute 23
36	Optional	Set/ Get		FieldMinCur	INT	Minimum Field Current for a DC Drive. Units: Amps / $2^{\text{CurrentScale}}$ where CurrentScale is attribute 23
37	Optional	Set	NV	Process Data Units	ENGUNITS	Applies to: ProcessActual ProcessRef Values specified in Volume I Appendix D.
38	Optional	Set	NV	Speed Control	BYTE	Set bits are used to request speed commands. See Semantics section.

AC/DC Drive Object, Class Code: 2A<sub>Hex</sub>

ttr ID	Need in Implem	Access Rule	NV	Name	Data Type	Description of Attribute
39	Optional	Get	V	Speed Status	BYTE	Set bits indicate the current speed status. see Semantics section.
40	Optional	Set	NV	Speed Trip Time	UINT	Time for SpeedActual beyond vendor specific hysteresis band. See Behavior section. Units: ms / $2^{\text{TimeScale}}$ where TimeScale is attribute 28
41	Optional	Get	NV	Max Rated Speed	INT	Vendor Specific maximum rating. Units: RPM / $2^{\text{MRSScale}}$ where MRSScale is attribute 46
42	Optional	Set	NV	Max Rated Speed Scale	SINT	Speed Scaling for the Maximum Rating. Scaling is accomplished as follows: ScaledSpeed = RPM / $2^{\text{MRSScale}}$ Range: -128 .. 127
43	Optional	Set	NV	SpeedStandby	INT	Speed setting for Standby. See Behavior section. Units: RPM / $2^{\text{SpeedScale}}$ where SpeedScale is attribute 22
44	Optional	Set	NV	SpeedActual Data Units	ENGUNITS	Allows a change from RPM RPM [default] Applies to: SpeedActual SpeedStandby Values specified in Volume I Appendix D
45	Optional	Set	NV	SpeedRef Data Units	ENGUNITS	Allows a change from RPM RPM [default] Applies to: SpeedRef Max Rated Speed LowSpdLimit HighSpdLimit Values specified in Volume I Appendix D.
46	Optional	Get	NV	Drive On Hours	DINT	Number of hours SpeedActual > 0 Value shall not "roll over" at terminal count

\* See section 5-30.5.1. for scaling example.

The basic units used by the AC/DC Drive Object are:

**Table 5-30.3 Basic Units**

Physical Quantity	Units	
Speed:	RPM	Revolutions Per Minute
Current:	100ma	100 milliAmps
Torque:	Nm	Newton Meters
Process control:	%	Percent
Power:	W	Watts
Voltage:	V	Volts
Time:	ms	Milliseconds

### 5-30.2.1 Semantics

#### 5-30.2.1.1 Speed Control and Speed Status

The *Speed Control* attribute is used to request various commands for the speed objective of the AC/DC Drive object. The value of the *Speed Status* attribute indicates the current speed status. See Behavior section for more information. Setting a bit to TRUE effects the request for the respective speed. A request to set the *Speed Control* attribute to an unsupported bit pattern (i.e., representing a command that the object does not support) shall yield an Invalid Attribute Value error response.

**Table 5-30.4 Speed Control Attribute and Speed Status Attribute Bit Map**

Bit	Speed Control	Speed Status	Status Description
0	Run Request	Running	On and SpeedActual > 0
1	Idle Request	At Idle	Zero current
2	Standby Request	At Standby Speed	SpeedActual = SpeedStandby
3	Coast Request	Coasting	Zero Torque
4	n.a. *	Stopped	SpeedActual = 0
5	n.a. *	Accelerating	SpeedActual is increasing
6	n.a. *	At Reference **	SpeedActual = SpeedRef
7	n.a. *	Decelerating	SpeedActual is decreasing

\* value shall be ignored

\*\* corresponds to the *AtReference* attribute (ID 3).

When these attributes (Speed Control and Speed Status) are supported by a device with a Control Supervisor object, *Speed Control* Bit 0 corresponds to *Run1* (Control Supervisor instance attribute ID 3) and *Speed Status* Bit 0 corresponds to *Running1* (Control Supervisor instance attribute ID 7).

Some implementations may support only some values of *Speed Status*. For example, At Idle and Stopped may always be set together (i.e., no difference between these two status conditions). A detailed description of the actual behavior corresponding to these speed conditions is vendor specific.

### 5-30.3 Common Services

**Table 5-30.4 AC/DC Drive Object Common Services**

Service Code	Need in Implementation		Service Name	Description of Service
	Class	Instance		
0E <sub>hex</sub>	Conditional*	Required	Get_Attributes_Single	Returns the contents of the specified attribute.
10 <sub>hex</sub>	n/a	Required	Set_Attributes_Single	Modifies an attribute value.
15 <sub>hex</sub>	n/a	Optional	Restore	Restores attribute values from a storage location accessible by the Save service. This service is only available if the drive is in the <b>Ready</b> or <b>NOT_READY</b> states.
16 <sub>hex</sub>	n/a	Optional	Save	Saves all attribute values to a location accessible by the Restore service.

See Appendix A for definition of these services

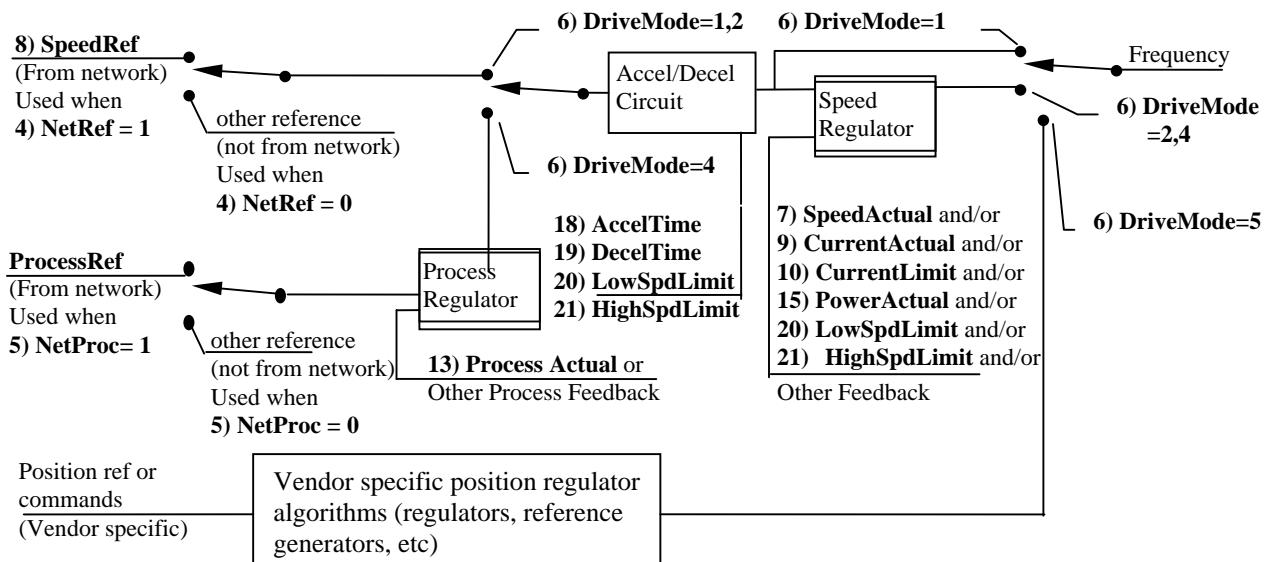
## 5-30.4 Object-specific Services

The AC/DC Drive object provides no object specific services.

## 5-30.5 Behavior

The following drawing represents the signal flow in an AC drive whose output is an inverter frequency command. Signal flow is controlled by the AC/DC Drive Instance attributes shown in **Bold** type. The process and speed references can be directed to originate from network commands or from references internal to the drive or supplied to the drive through terminal blocks or keypads. Network control of these alternative reference sources will be vendor specific. The mode of operation can also be controlled from the network with the mode attribute AC/DC Drive object. Note that not all modes will be supported by all drives.

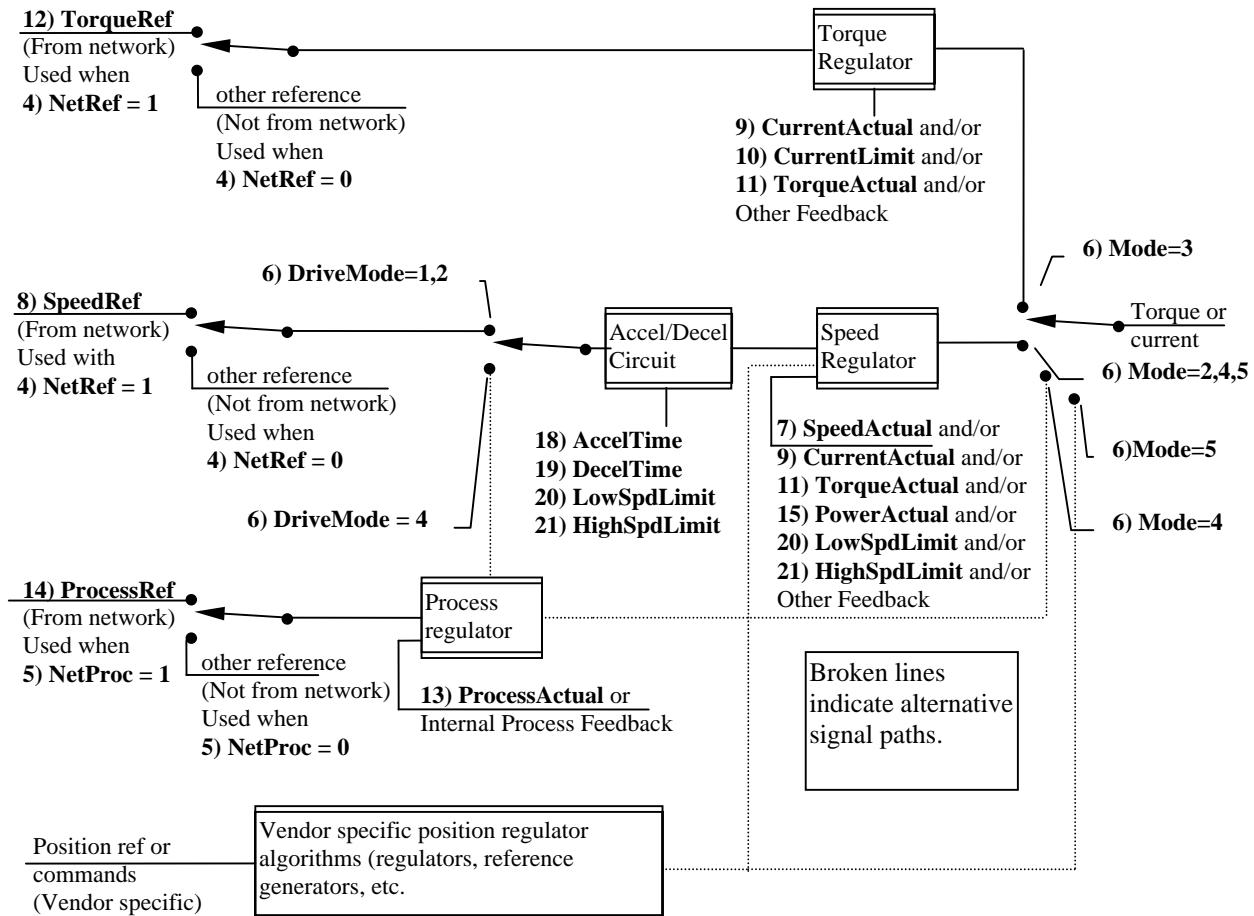
**Table 5-30.5 AC/DC Drive Object Behavior - Frequency Inverter**



The following drawing represents the signal flow for a AC or DC torque or current controlled drive. These drives can operate in Mode 3, allowing a network torque reference to directly control the drive.

Each drive vendor can implement a process regulator and position regulator to feed either the speed regulator or the torque command directly. Broken lines indicate these alternative signal paths.

Table 5-30.6 AC/DC Drive Object Behavior – Torque or Current Controlled

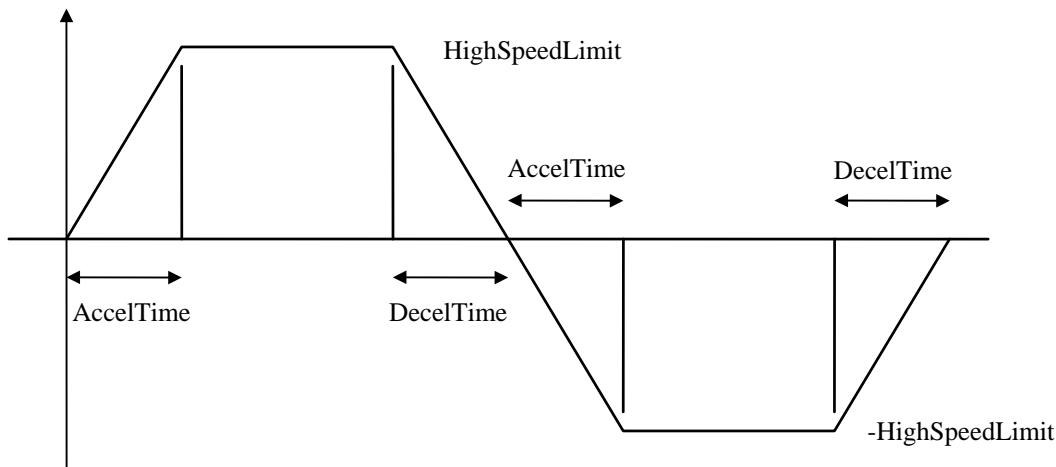


As illustrated below, **AccelTime** (attribute 18) is defined as the time in seconds it would take the drive to accelerate from zero to **HighSpdLimit** (attribute 21), the maximum speed at which the drive is allowed to operate.

Similarly, **DecelTime** (attribute 19) is the time to slow down from **HighSpdLimit** (attribute 21) to zero speed.

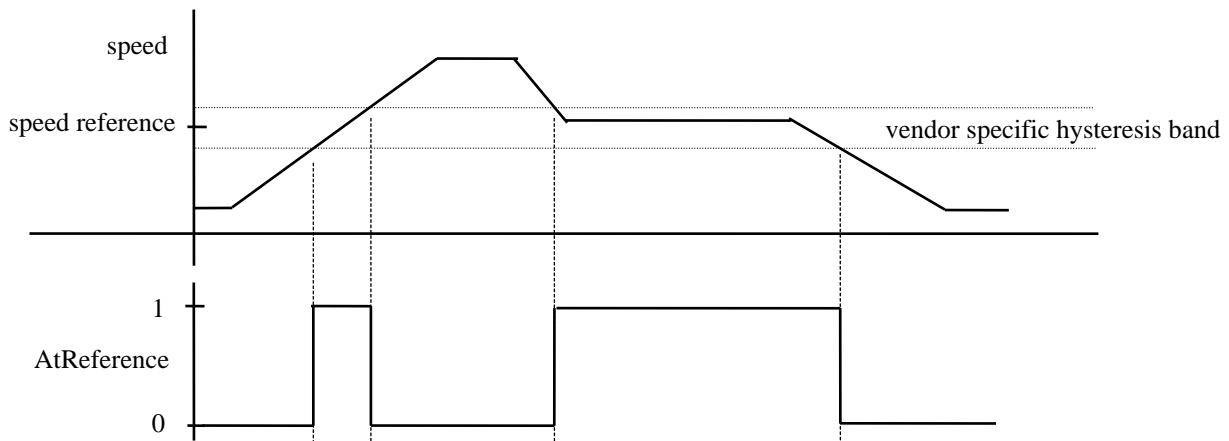
There will be no guarantee that the drive will operate at these rates in actual applications, since current limit may increase the acceleration time, and lack of sufficient braking capability may increase the stopping time.

Figure 5-30.7 Accel/Decel Time



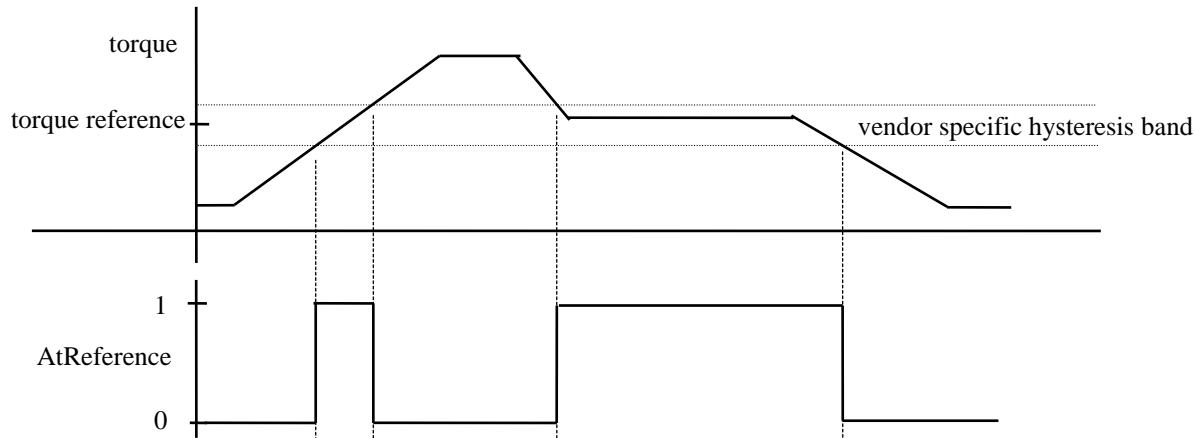
When **DriveMode** (attribute 6) is set to “1” or “2”, **AtReference** (attribute 3) is “1” when **SpeedActual** (attribute 7) is at its prescribed speed reference plus or minus a vendor specific hysteresis band. The vendor specific hysteresis band may be fixed, programmable, or zero.

Figure 5-30.8 Vendor Specific Hysteresis Band – Speed Reference



When **DriveMode** (attribute 6) is set to “3”, **AtReference** (attribute 3) is “1” when **TorqueActual** (attribute 11) is at its prescribed torque reference plus or minus a vendor specific hysteresis band. The vendor specific hysteresis band may be fixed, programmable, or zero.

**Figure 5-30.9 Vendor Specific Hysteresis Band – Torque Reference**



### 5-30.5.1 Enhanced Speed Control Behavior

The *Speed Control* attribute is used to request different speed objectives for the AC/DC Drive object. With no bits set to True, the objective for the object is to obtain the speed specified by the value of the *SpeedRef* attribute. Other Speed Control bits are used to request alternative speed objectives, including: idle, standby and coast.

For DriveMode=1,2, the Standby alternative speed objective alters the source of the speed setpoint (input to Accel/Decel Circuit) from *SpeedRef* (attribute 8) to *SpeedStandby* (attribute 43).

Irrespective of DriveMode; idle and coast affect the appropriate action by overriding the Accel/Decel Circuit. *SpeedRef* and *SpeedStandby* are ignored by the Accel/Decel Circuit while idle or coasting.

The *Speed Trip Time* attribute specifies a time for which the value of *SpeedActual* may exceed the vendor specific hysteresis band before an Speed Trip Condition exists. The particular actions and/or behavior modifications associated with an Speed Trip Condition shall be specified by the Device Profile.

### 5-30.5.2 Scaling of attribute values

As part of the AC/DC Drive Object definition, engineering units are defined for each physical quantity, e.g. RPM for Velocity, Nm for Torque etc. To maximize the resolution capable or necessary on some devices or applications, these values can be normalized using a binary scale factor before transmission on the bus. A separate scaling factor is specified for each physical quantity. Normally, scaling factors will be set up once during initialization according to the range of values to be used in the application.

Scaling Factors allow the representation of physical units on the bus to obtain an acceptable resolution and dynamic range for all applications.

**Example:** Configuration of a DC Drive to operate with RPM resolution of 0.125 RPM

$$\begin{aligned} \text{SpeedRef (AC/DC Drive Object, Attribute ID 8)} &= 4567 \\ \text{SpeedScale (AC/DC Drive Object, Attribute ID 22)} &= 3 \\ \Rightarrow \text{Actual Commanded Speed} = \text{SpeedRef} / 2^{\text{SpeedScale}} &= 4567 / 2^3 = 570.875 \text{ RPM} \end{aligned}$$

Input from Drive to bus:

$$\begin{aligned} \text{Actual Drive Operating Speed} &= 789.5 \text{ RPM} \\ \text{SpeedScale (AC/DC Drive Object, Attribute ID 22)} &= 3 \\ \Rightarrow \text{SpeedActual (AC/DC Drive Object, Attribute ID 7)} &= \\ \text{Actual Operating Speed} \times 2^{\text{SpeedScale}} &= 789.5 \times 2^3 = 6316. \end{aligned}$$

In cases where the applicable scaling factor attribute is non-zero, the units are:

$$\text{Engineering unit} / 2^{\text{Scale Factor Attribute}}$$

In the above example, therefore, the units are 0.125 RPM.

Drives that do not support scale factors (Attributes 22 ~ 28), should return an “Attribute Not Supported” error. In this case, the default scaling is presumed, and all physical quantities will then default to engineering units. For example, speed values will then default to units of RPM. Drives that do not use RPM as an internal measure should calculate an RPM value based on motor parameters. For example, AC VFD’s may calculate an RPM/Hz constant using motor rated frequency and motor base speed from the Motor Data Object. An example of this type of implementation (including scaling factors) for the commanded VFD’s frequency may be:

$$\text{Frequency Command (Hz)} = (\text{SpeedRef} \times \text{RatedFreq}) / (2^{\text{SpeedScale}} \times \text{BaseSpeed})$$

and an example implementation for the reported operating speed as a function of the output frequency may be:

$$\text{SpeedActual} = \text{Int}[(\text{Output Frequency (Hz)} \times \text{BaseSpeed} \times 2^{\text{SpeedScale}}) / \text{RatedFreq}]$$

Where:

Frequency Command = the VFD’s resultant frequency command, the resolution of which is vendor-dependent.

SpeedRef = AC/DC Drive Object Instance Attribute ID 8.

RatedFreq = Motor Data Object, AC Motor Instance Attribute ID 9.

SpeedScale = AC/DC Drive Object Instance Attribute ID 22.

BaseSpeed = Motor Data Object, AC Motor Instance Attribute ID 15.

SpeedActual = AC/DC Drive Object Instance Attribute ID 7.

Int = A vendor-specific integer typecast function (such as rounding or truncation).

Output Frequency = the VFD’s reported present operating frequency, the resolution of which is vendor-dependent.

## 5-31 Acknowledge Handler Object

**Class Code: 2B<sub>hex</sub>**

The Acknowledge Handler Object is used to manage the reception of message acknowledgments. This object communicates with a message producing Application Object within a device. The Acknowledge Handler Object notifies the producing application of acknowledge reception, acknowledge timeouts, and production retry limit.

### 5-31.1 Class Attributes

**Figure 5-31.1 Acknowledge Handler Object Class Attributes**

Number	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1 thru 7	These class attributes are optional and are described in Chapter 4 of this specification.					

### 5-31.2 Class Services

- Get\_Attribute\_Single
- Create
- Delete

**Figure 5-31.2 Acknowledge Handler Object Common Services - Class Level**

Need in Implementation	Name	Service Code	Description of Service
Optional	Get_Attribute_Single	0Ehex	Used to read an Acknowledge Handler Object attribute value.
Optional	Create	08hex	Used to create an Acknowledge Handler Object.
Optional	Delete	09hex	Used to delete all dynamically created Acknowledge Handler Object instances.

See Appendix A for definitions of these services

### 5-31.3 Instance Attributes

Figure 5-31.3 Acknowledge Handler Object Instance Attributes

Attr ID	Need in Implement	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1	Required	Set	Acknowledge Timer	UINT	Time to wait for acknowledge before resending	Range 1-65,535 ms (0 invalid) default = 16
2	Required (Get) Optional (Set)	Get/Set	Retry Limit	USINT	Number of Ack Timeouts to wait before informing the producing application of a RetryLimit_Reached event.	Range 0-255 default = 1
3	Required	Set (Inactive) Get (Active)	COS Producing Connection Instance	UINT	Connection Instance which contains the path of the producing I/O application object which will be notified of Ack Handler events.	Connection Instance ID
4	Optional	Get	Ack List Size	BYTE	Maximum number of members in Ack List	0 = Dynamic >0 = Max number of members
5	Optional	Get	Ack List	BYTE Array of UINT	List of active connection instances which are receiving Acknowledgments.	Number of members followed by list of: Connection Instance ID
6	Optional	Get	Data with Ack Path List Size	BYTE	Maximum number of members in Data with Ack Path List	0 = Dynamic >0 = Max number of members
7	Optional	Get	Data with Ack Path List	BYTE Array of UINT USINT Padded EPATH	List of connection instance/consuming application object pairs. This attribute is used to forward data received with acknowledgment.	Number of members followed by list of: Connection Instance ID/CIP path length/CIP path

- If the specified value for the Acknowledge Timer attribute is not equal to an increment of the available clock resolution, then the value is rounded up to the next serviceable value. For example: a Set\_Attribute\_Single request is received specifying the value 5 for the Acknowledge Timer attribute and the product provides a 10 millisecond resolution on timers. In this case the product would load the value 10 into the Acknowledge Timer attribute.
- The value that is actually loaded into the Acknowledge Timer attribute is reported in the Service Data Field of a Set\_Attribute\_Single response message associated with a request to modify this attribute.
- A successful set attribute to the Retry Limit attribute will reset the Retry Counter.

**Acknowledge Handler Object, Class Code: 2B<sub>Hex</sub>**

- The Ack List attribute is updated when an associated connection transitions between configuring, established, timed-out, and non-existent. Refer to the State Event Matrix for details.
- If the Acknowledge Handler Object is active (at least one member in the Ack List), the COS Producing Connection Instance attribute access is get only.
- The default value loaded into the Acknowledge Timer attribute at time of instantiation is 16 ms.
- The default value loaded into the Retry Limit attribute at time of instantiation is 1.

**5-31.4 Instance Services**

The Acknowledge Handler Object Instance supports the following Common Services:

- Get\_Attribute\_Single
- Set\_Attribute\_Single
- Delete

**Figure 5-31.4 Acknowledge Handler Object Common Services – Instance Level**

Need in Impl	Name	Service Code	Description of Service
Required	Get_Attribute_Single	0Ehex	Used to read an Acknowledge Handler Object attribute value.
Required	Set_Attribute_Single	10hex	Used to modify an Acknowledge Handler Object attribute value.
Optional	Delete	09hex	Used to delete an Acknowledge Handler Object.

See Appendix A for definition of these services

**5-31.5 Object-specific Services**

The Acknowledge Handler Object Instance also supports the following Object Class Specific Services:

- Add\_AckData\_Path
- Remove\_AckData\_Path

**Figure 5-31.4 Acknowledge Handler Object Object-specific Services**

Need in Impl	Name	Service Code	Description of Service	Service Request Parameters
Optional	Add_AckData_Path	4Bhex	Adds a path for data with acknowledgment for a connected consumer.	Connection Instance CIP Path Length CIP Path
Optional	Remove_AckData_Path	4Chex	Removes a path with acknowledge path for the given connected consumer.	Connection Instance

These services are used to add and remove paths for each of the connected acknowledge consumers when data is sent with the acknowledgment.

## 5-31.6 Behavior and Configuration of Acknowledged Data Production

The following rules are used to configure and determine the behavior of an acknowledged Change of State or Cyclic I/O connection using the Acknowledge Handler Object. In the following examples, COS Producer is used to reference the device producing change of state or cyclic data and consuming an acknowledgment (client). COS Consumer is used to reference the device consuming the change of state or cyclic data and producing an acknowledgment (server).

### 5-31.6.1 Acknowledged Data Production

1. The COS Producers consumed connection path must be set to an available Acknowledge Handler Object. The path must consist of Class and Instance. If an Acknowledge Handler Object is not available, use the Acknowledge Handler Class Create service to obtain a new one.
2. The COS Producers producing I/O application informs the Acknowledge Handler Object of new data production (Data Sent event message) or data production retries (Data Resent event message).
3. The COS Producers acknowledgment reception is done by the Acknowledge Handler Object. The Acknowledge Handler object informs the Producing I/O application when one or more acknowledges have not been received within the acknowledge timeout (using the Ack List and Ack Timeout attributes).
4. The acknowledge message requires no data. The COS producing device's Acknowledge Handler object should consider valid message reception as an acknowledgment. However, a change of state or cyclic producing device may be configured to consume data along with the acknowledgment. In this case the data is forwarded to the application object in the 'Data with Ack Path List' attribute, based on the connection which received the data.
5. The COS Consumer acknowledge producing application should be configured to send either a zero length message or valid response (output) message when a valid input message is consumed.
6. An acknowledge timer is started each time production occurs. The Acknowledge Handler object is notified of this event by a Data Sent or Data Resent event message from the producing application.
7. Expiration of the acknowledge timer causes an Acknowledge Timeout message to be sent to the producing application object. That object must resend the last message if the Retry Limit has not been reached. It may also take an application specific action.
8. The retry count is incremented each time an Acknowledge Timeout message is sent to the producing application. When the retry limit has been reached, a Retry Limit Reached message is sent to the producing application object.
9. The retry count is cleared on each Data Sent message. A Data Resent message does *not* clear the retry counter.
10. The acknowledge timer value is configurable within the Acknowledge Handler object.

**Acknowledge Handler Object, Class Code: 2B<sub>Hex</sub>**

11. The number of retries is (optionally) configurable within the Acknowledge Handler object.

### **5-31.6.2 Change of State Examples**

#### **5-31.6.2.1 Acknowledged Change of State (Using one connection object and one COS consumer)**

For the producer:

1. Create a Connection Object.
2. The producer transportClass\_trigger attribute is set to Class 2/3, Change-Of-State, Client (13h for Class 3) or Class2/3, Cyclic, Client (03h for Class 3).
3. The produced connection path is set to the producing application, which must support Change of State or Cyclic production.
4. Create an Acknowledge Handler object. Configure the 'COS Producing Connection Instance' attribute to the instance of the connection object which specifies the producing I/O application object. The Ack List attribute will be updated with the connection instance of the I/O connection object which is consuming an acknowledge as that connection transitions to the established state. Optionally configure the Acknowledge Timer and Retry Count attributes.
5. The consumed connection path is set to the Acknowledge Handler object just created and configured for handling the acknowledge. The path contains Class and Instance only.
6. The consumed connection size is set to the size of the data expected to be delivered to the object specified in the consumed path of the Acknowledge Handler Object, or zero if no path is configured.
7. All other attributes are configured the same way as any other peer to peer connection.

For the consumer:

8. The consumer transportClass\_trigger attribute is set to Class 2/3, Server (83h for Class 3).
9. The produced connection path is set to the consuming application (or some other application, this is product specific) for generating the acknowledge.
10. The produced connection size is set to the size of the data to be delivered to the object specified in the consumed path of the change of state producing device's Acknowledge Handler object, or zero if no path is configured.
11. All other attributes are configured the same way as any other peer to peer connection.

**5-31.6.2.2 Acknowledged Change of State (Using multiple connection objects and COS consumers)**

For the producer:

1. Create a Connection Object.
2. The producer transportClass\_trigger attribute is set to Class 0, Change-Of-State, Client (10h) or Class 0, Cyclic, Client (00h).
3. The produced connection path is set to the producing application which must support Change of State or Cyclic production.
4. The consumed connection path and consumed connection size are not configured.
5. All other attributes are configured the same way as any other peer to peer connection.
6. Create a Connection Object for each COS consumer.
7. The consumer transportClass\_trigger attribute is set to Class 0, Server (80h) for each consuming connection object.
8. Create an Acknowledge Handler object. Configure the 'COS Producing Connection Instance' attribute to the instance of the connection object which specifies the producing I/O application object. The Ack List attribute will be updated with the connection instance of each I/O connection object which is consuming an acknowledge as those connections transition to the established state. Optionally configure the Acknowledge Timer and Retry Count attributes.
9. The consumed connection path for each consuming connection object is set to the Acknowledge Handler object just created and configured for handling the acknowledge. The path contains Class and Instance only.
10. The consumed connection sizes are set to the size of the data expected to be delivered to the consumed path set in the Acknowledge Handler Object, or zero if no path is configured.
11. All other attributes are configured the same way as any other peer to peer connection.

For each consumer:

12. The consumer transportClass\_trigger attribute is set to Class 2/3, Server (83h for Class 3).
13. The produced connection path is set to the consuming application (or some other application, this is product specific) for generating the acknowledge.
14. The produced connection size is set to the size of the data to be delivered to the consumed path set in the change of state producing device's Acknowledge Handler object, or zero if no path is configured.
15. All other attributes are configured the same way as any other peer to peer connection.

**Acknowledge Handler Object, Class Code: 2B<sub>Hex</sub>**

Once the Connection Object(s) is (are) configured, an Apply Attributes service is sent to each configured Connection Object to transition the connection(s) to the Established state. During the apply, the connection object is 'delivered' to both the consuming and producing application objects for validation of the attribute information. At this time, the producing I/O application checks for a change of state configuration by examining the transportClass\_trigger attribute. If the Production Trigger bits are set to change-of-state or cyclic, the application will configure itself for change of state or cyclic production. If change of state or cyclic production is not supported by the producing application object an error must be returned to the apply\_attributes service.

**5-31.6.3 Use of Timers with Acknowledged Data Production**

The following rules must be observed when sending acknowledged data.

New data not sent while the Inhibit Timer is active (running).

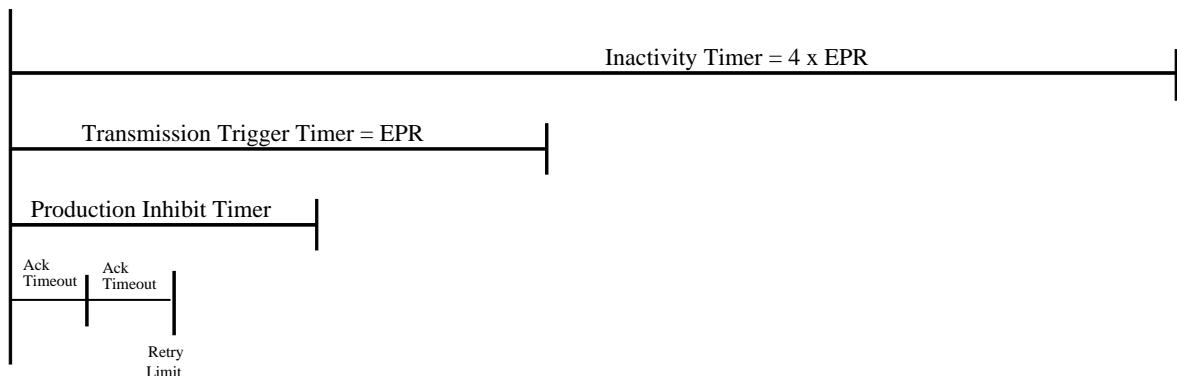
New data is sent when no acknowledge is pending, subject to rule # 1. (An acknowledge is pending after a send of new data or a retry of old data and until an Ack Timeout or Ack Received.

Retry of old data occurs at Ack Timeout if new data is not available or the Inhibit Timer is active.

Sending new data (or old data on transmission trigger timeout) starts the Ack Timer, Inhibit Timer, and the Transmission Trigger Timer. The Retry Counter is also cleared.

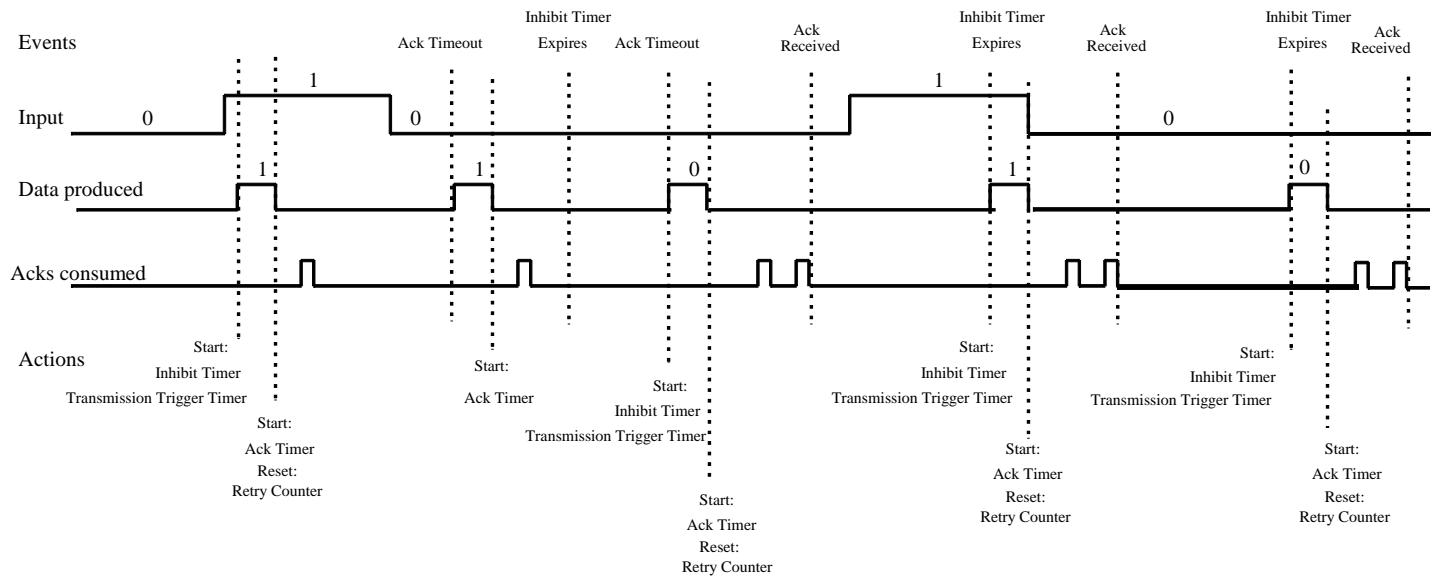
A retry of old data starts the Ack Timer.

**Figure 5-31.5 Typical Timing Relationships for Acknowledged Data Production**



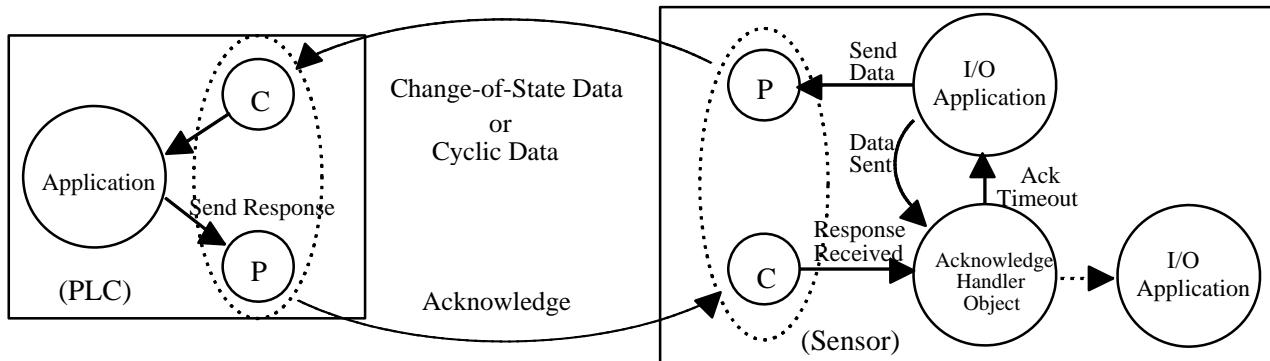
Acknowledge Handler Object, Class Code: 2B<sub>Hex</sub>

Figure 5-31.6 Example of a COS System, 2 Acking Devices



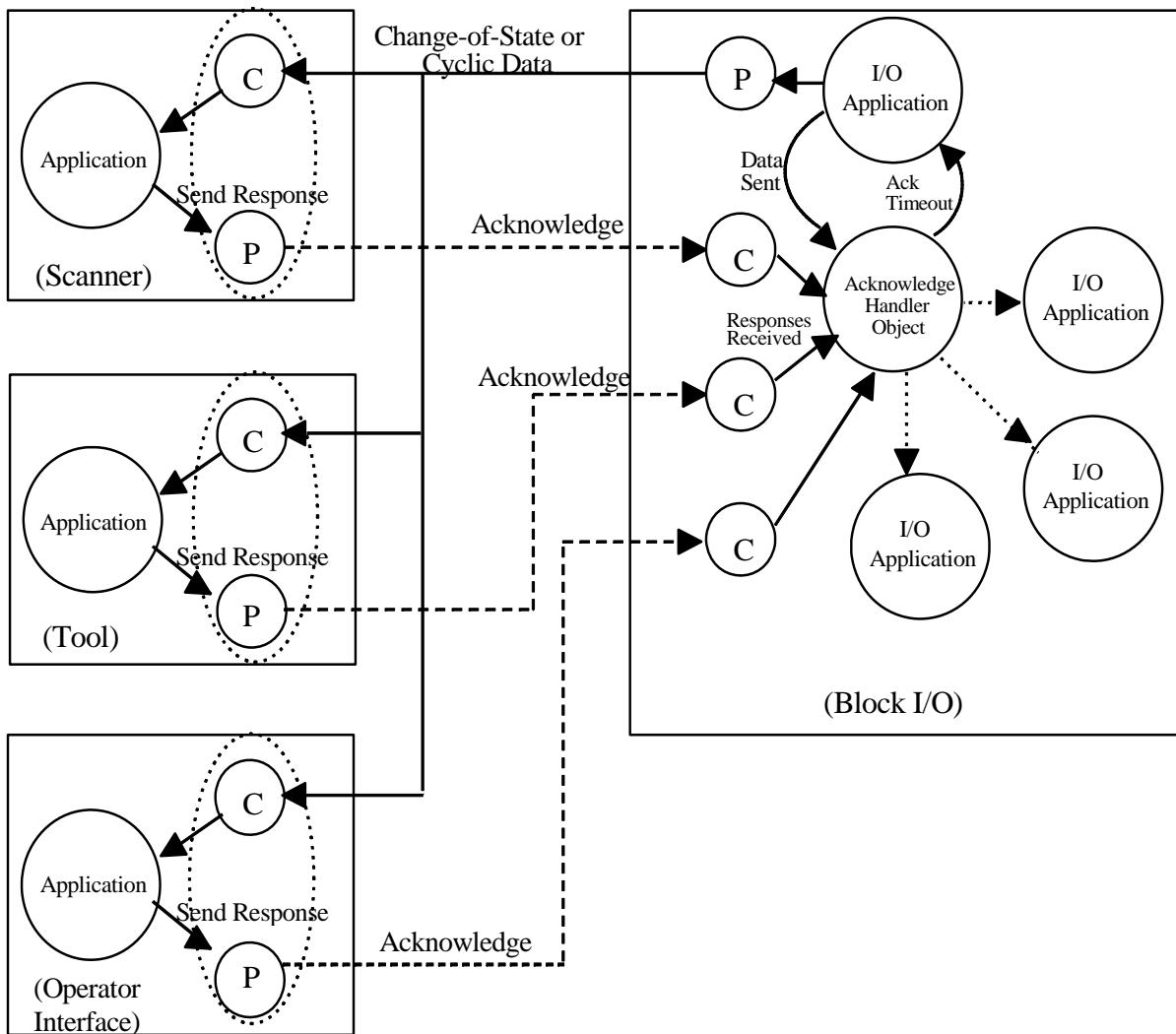
The following diagrams illustrate the message flow in a Change of State connection for both single and multi-consumer configurations.

Figure 5-31.7 Message Flow in COS Connection – One Connection Object, One Consumer



Acknowledge Handler Object, Class Code: 2B<sub>Hex</sub>

Figure 5-31.8 Message Flow in COS Connection –Multiple Consumers



**Acknowledge Handler Object, Class Code: 2B<sub>Hex</sub>**

The following are the State Event Matrices for the Producing I/O application object and Acknowledge Handler object associated with a Change of State connection.

**Figure 5-31.9 SEM for Producing I/O Application Object**

State ►	Not Running	Running	Running with Acknowledgment	Prohibited
Events ▼				
<b>Change of State Detected</b>	Ignore Event	Inform Link Producer to Send Data. If Inhibit Time configured Start Inhibit Timer Transition to Produced	Inform Link Producer to Send Data. Send DataSent event message to Acknowledge Handler object. If Inhibit Time configured Start Inhibit Timer transition to Prohibited.	Queue Event
<b>Acknowledge Received</b>	Not Applicable	Not Applicable	Ignore Event	If Ack_Active Set If Inhibit Timer not running Transition to Running with Acknowledgement Else Set Ack_Receive flag Else ignore event
<b>Acknowledge Timeout</b>	Ignore Event	Not Applicable	Inform Link Producer to Send Data Send Data_Resent event message to Acknowledge Handler Object.	Inform Link Producer to Send Data. Send Data_Resent event message to Acknowledge Handler Object.
<b>Transmission Timer Expires</b>	Not Applicable	Inform Link Producer to Send Data.	Inform Link Producer to Send Data. Send Data_Sent event message to Acknowledge Handler Object.	Inform Link Producer to Send LAST Data Sent. Send Data_Sent event message to Acknowledge Handler Object.
<b>Retry Limit Reached</b>	Ignore Event	Not Applicable	Product Specific	Transition to Running with Acknowledgement.
<b>Inhibit timer Expires</b>	Ignore Event	Not Applicable	Not Applicable	If Ack_Active set If Ack_Received Transition to Running w/Ack Clear Ack_Received flag Else Ignore Event Else Transition to Running
<b>Connection Deleted</b>	Not Applicable	Transition to Not Running	Transition to Not Running	Transition to Not Running
<b>Acknowledge Active</b>	Set Ack Active Flag	Transition to Running with Acknowledgement Set Ack Active Flag	Ignore Event	Set Ack_Active Flag
<b>Acknowledge Inactive</b>	Reset Ack Active Flag	Ignore Event	Transition to Running Reset Ack Active Flag	Reset Ack_Active Flag
<b>Connection Transition to Established</b>	If Ack Active Transition to Running with Acknowledgment If Ack Inactive Transition to Running	Ignore Event	Ignore Event	Ignore Event

Note: This is a partial SEM for a producing I/O application object. Only those states and events associated with data acknowledgement are defined. Other states and events will most likely be associated with a producing I/O application object.

**Acknowledge Handler Object, Class Code: 2B<sub>Hex</sub>****Figure 5-31.10 SEM for Acknowledge Handler Object**

<b>State ►</b>	<b>Non-Existent</b>	<b>Inactive</b>	<b>Active</b>
<b>Events ▼</b>			
<b>Receive Acknowledge</b>	Not Applicable	Ignore Event	Clear Ack Flag Forward any data to application object If all Acknowledges received Clear Ack Timer and Retry Counter Send Acknowledge_Received event message to producing application object
<b>Acknowledge Timer Expires</b>	Not Applicable	Ignore Event	Send Ack_Timeout event message to producing application object.
<b>Data Sent</b>	Not Applicable	Ignore Event	Set Ack Required Flag Set Ack Timer Clear Retry Counter
<b>Data Resent</b>	Not Applicable	Ignore Event	Set Ack Required Flag & Ack Timer If Retry_Limit > 0 Increment Retry Counter If Retry Counter = Retry_Limit Send Rety_Limit_Reached event message to producing application object
<b>Delete</b>	Not Applicable	Transition to Non-Existent	Transition to Non-Existent
<b>Create</b>	Transition to Inactive	Not Applicable	Not Applicable
<b>Apply Attributes</b>	Not Applicable	Verify new connection can be added to list. Pass this message to the consumed application object, if one is configured for this connection.	Verify new connection can be added to list. Pass this message to the consumed application object, if one is configured for this connection.
<b>Connection Transition to Established</b>	Not Applicable	Add this connection instance to the connection list (or internally flag as 'Acking'). Pass this message to the consumed application object, if one is configured for this connection. Send Acknowledge_Active event message to producing application. Transition to Active.	Add this connection instance to the connection list. Pass this message to the consumed application object, if one is configured for this connection.
<b>Inactivity/Watchdog Timer Expires</b>	Not Applicable	Not Applicable	Internally flag this connection as 'Not Acking'. An acknowledgement will no longer be monitored for this connection, however it remains in the connection list. Pass this event to the consumed application object, if one is configured for this connection. If no 'Acking' connections in list, send Acknowledge_Inactive event to producing application and transition to Inactive.
<b>Connection Deleted</b>	Not Applicable	Ignore Event	Remove this connection instance from the connection list. Pass this event to the consumed application object, if one is configured for this connection. If no 'Acking' connections in list, send Acknowledge_Inactive event to producing application and transition to Inactive.

## 5-32 Overload Object

**Class Code: 2C<sub>hex</sub>**

This object models all the functions specific to an AC motor overload protection device.

### 5-32.1 Class Attributes

**Table 5-32.1 Overload Object Class Attributes**

Attribute ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute
1 thru 7	These class attributes are optional and are described in Chapter 4 of this specification.				

### 5-32.2 Instance Attributes

**Table 5-32.2 Overload Object Instance Attributes**

Attribute ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute
1	Optional	Get	NumAttr	USINT	Number of Attributes supported
2	Optional	Get	Attributes	Array of USINT	List of Attributes supported
3	Optional	Set	TripFLCSet	INT	Overload Full Load Current Setting Units : 100ma / 2 <sup>CurrentScale</sup> where CurrentScale is attribute 12
4	Optional	Set	TripClass	USINT	Trip Class Setting 0 to 200
5	Optional	Get	AvgCurrent	INT	Average of the three phase current. Units: 100ma / 2 <sup>CurrentScale</sup> where CurrentScale is attribute 12
6	Optional	Get	%PhImbal	USINT	% Phase Imbalance
7	Optional	Get	%Thermal	USINT	% Thermal Capacity
8	Optional	Get	CurrentL1	INT	Actual motor phase current L1 Units: 100ma / 2 <sup>CurrentScale</sup> where CurrentScale is attribute 12
9	Optional	Get	CurrentL2	INT	Actual motor phase current L2 Units: 100ma / 2 <sup>CurrentScale</sup> where CurrentScale is attribute 12
10	Optional	Get	CurrentL3	INT	Actual motor phase current L3 Units: 100ma / 2 <sup>CurrentScale</sup> where CurrentScale is attribute 12
11	Optional	Get	GroundCurrent	INT	Ground Current Units: 100ma / 2 <sup>CurrentScale</sup> where CurrentScale is attribute 12
12	Optional	Set	CurrentScale	SINT	Current scaling factor. Scaling is accomplished as follows: ScaledCurrent = 100ma / 2 <sup>CurrentScale</sup> Range: -128 .. 127

### 5-32.2.1 Scaling of attribute values

As part of the Overload object definition, a separate scaling factor is specified current. To maximize the resolution capable or necessary on some devices or applications, current values can be normalized using a binary scale factor before transmission on the bus. Normally, this scaling factor will normally be set up once during initialization depending on the range of current values to be used in the application.

CurrentScale allow the representation of current on the bus to be kept as short as possible while still preserving an acceptable resolution and dynamic range for all applications.

#### Example:

Output (to device from bus):

TripFLCSet(Overload Object, Attribute ID 3) = 23456

CurrentScale (Overload Object, Attribute ID 12) = 8

⇒ Actual FLC setting used by device = TripFLCSet =  $23456 \div 2^8 = 91.625 \times 100\text{mA} = 9.1625 \text{ amps}$

Input (from drive to bus):

Actual FLC setting used by device = 78.95 amps =  $789.5 \times 100\text{mA}$

CurrentScale (Overload Object, Attribute ID 12) = 3

⇒ TripFLCSet(Overload Object, Attribute ID 3) =  $789.5 \times 2^8 = 6316$

Devices that do not support CurrentScale (Attributes 12), should return a zero, or not supported attribute error. In this case, the default scaling (0) is presumed. All currents will then default to units of 100ma.

### 5-32.3 Common Services

Table 5-32.3 Overload Object Common Services

Service Code	Need in Implementation		Service Name	Description of Service
	Class	Instance		
0E <sub>hex</sub>	Optional	Required	Get_Attributes_Single	Returns the contents of the specified attribute.
10 <sub>hex</sub>	n/a	Required	Set_Attributes_Single	Modifies an attribute value.
15 <sub>hex</sub>	n/a	Optional	Restore	Restores attribute values from a storage location accessible by the Save service. This service is only available if the drive is in the <b>Ready_Disabled</b> or <b>Wait_Power</b> states.
16 <sub>hex</sub>	n/a	Optional	Save	Saves all attribute values to a location accessible by the Restore service.

See Appendix A for definition of these services

### 5-32.4 Object-specific Services

The Overload object provides no object specific services.

### **5-32.5 Behavior**

The Overload object behavior can be changed by setting attributes, overload current setting and Trip Class setting will effect the trip curve of the device.

## 5-33 Softstart Object

**Class Code: 2D hex**

This object models all the functions specific to a Soft Start Motor Starter.

### 5-33.1 Class Attributes

**Table 5-24.1 Softstart Object Class Attributes**

Attribute ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute
1 thru 7	These class attributes are optional and are described in Chapter 4 of this specification.				

### 5-33.2 Instance Attributes

**Table 5-24.2 Softstart Object Instance Attributes**

Attribute ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute
1	Optional	Get	NumAttr	USINT	Number of Attributes supported
2	Optional	Get	Attributes	Array of USINT	List of Attributes supported
3	Required	Get	AtReference	BOOL	Starting/stopping output voltage reference status 0 = Not At Reference 1 = Output At Voltage Reference
4	Required	Set	StartMode	USINT	0 = No Voltage Ramp No Current Limit 1 = Voltage Ramp No Current Limit 2 = No Voltage Ramp Current Limit 3 = Voltage Ramp Current Limit 4 – 9 = Reserved by CIP 10 – 255 = Vendor Specific
5	Optional	Set	StopMode	USINT	0 = Coast 1 = Ramp Down 2 = Brake 3 – 9 = Reserved by CIP 10 – 255 = Vendor Specific
6	Optional	Get/Set	RampMode	BYTE	0 = Single Ramp 1 = Dual Contiguous Ramp 2 = Dual Independent Ramp 3 – 9 = Reserved by CIP 10 – 255 = Vendor Specific
7	Optional	Get/Set	RampTime1	UINT	Tenths of Seconds
8	Optional	Get/Set	InitialVoltage1	USINT	0 - 100 % Of Full Line Voltage
9	Optional	Get/Set	RampTime2	UINT	Tenths of Seconds
10	Optional	Get/Set	InitialVoltage2	USINT	0 - 100 % of Full Line Voltage
11	Optional	Get/Set	Rotation	BOOL	0 = ABC phase rotation 1 = CBA phase rotation
12	Optional	Get/Set	KickStart	BOOL	0 = disabled 1 = enabled fixed time

**Softstart Object, Class Code: 2D<sub>Hex</sub>**

<b>Attribute ID</b>	<b>Need in Implementation</b>	<b>Access Rule</b>	<b>Name</b>	<b>Data Type</b>	<b>Description of Attribute</b>
13	Optional	Get/Set	KickStartTime	USINT	Tenths of Seconds
14	Optional	Get/Set	KickStartVoltage	UINT	0 - 100 % of Full Line Voltage
15	Optional	Get/Set	EnergySaver	BOOL	0 = disabled 1 = enabled
16	Optional	Get/Set	DecelTime	UINT	Tenths of Seconds
17	Optional	Get/Set	CurrentLimitSet	UINT	Percent of Rated Current (Motor Data Instance Attribute 6)
18	Optional	Get/Set	BrakingCurrentSet	UINT	Percent of Rated Current (Motor Data Instance Attribute 6)

**5-33.3 Common Services****Table 5-24.3 Softstart Object Common Services**

<b>Service</b>	<b>Need in Implementation</b>		<b>Service Name</b>	<b>Description of Service</b>
	<b>Code</b>	<b>Class</b>		
0E <sub>hex</sub>	Optional	Required	Get_Attributes_Single	Returns the contents of the specified attribute.
10 <sub>hex</sub>	n/a	Required	Set_Attributes_Single	Modifies an attribute value.
15 <sub>hex</sub>	n/a	Optional	Restore	Restores attribute values from a storage location accessible by the Save service. This service is only available if the drive is in the <b>Ready_Disabled</b> or <b>Wait_Power</b> states.
16 <sub>hex</sub>	n/a	Optional	Save	Saves all attribute values to a location accessible by the Restore service.

See Appendix A for definition of these services

**5-33.4 Object-specific Services**

The Softstart object provides no object specific services.

**5-33.5 Behavior****5-33.5.1 Starting Behavior**

The controlled starting of motors can be accomplished using one or more optional approaches; Voltage Ramp (single ramp, dual contiguous ramp, or dual independent ramp options), Current Limit, or a combination of Voltage Ramp /Current Limit.

StartMode (Attribute 4) configures the device to Voltage Ramp / Current Limit starting behavior.

**5-33.5.1.1 Voltage Ramp Starting**

Voltage Ramp based starting (with no current limiting) is selected by setting the StartMode (Attribute 4) to a value of “1”. The mode in which the voltage is ramped up is selected by RampMode (Attribute 6).

**Softstart Object, Class Code: 2D<sub>Hex</sub>**

When RampMode (Attribute 6) = 0, Single Ramp mode is selected. In this mode the output voltage of the Soft Starter is increased at a fixed rate, over a user defined time period, RampTime1 (Attribute 7), until it reaches the full line voltage rating of the softstart. To overcome motor/load inertia, the output voltage may be set to higher than zero voltage, InitialVoltage1 (Attribute 8) before the voltage ramp up is started. For very high load inertia situations, the full line voltage can be applied for a fixed period of time: KickStart (Attribute 13) is set to 1, or optionally a user defined voltage (specified as a % of full line voltage) for a user defined period of time: Kick Start (Attribute 13) is set to 1, KickStartTime (Attribute 12) (greater than zero), and Kick Start Voltage (Attribute 18) (greater than zero) values are entered.

When RampMode (Attribute 6) = 1, Dual Contiguous Ramp mode is selected. In this mode the output voltage is increased at a fixed rate, over a user defined time period, RampTime1 (Attribute 7), until it reaches InitialVoltage2 (Attribute 10). The output voltage then increases at a different fixed rate, over a second user defined time period, RampTime2 (Attribute 9), until it reaches full line voltage. As in single ramp soft starters, the first ramp of a dual contiguous ramp soft starter may be preceded with Initial Voltage1 and/or Kick Start functions.

When RampMode (Attribute 6) = 2, Dual Independent Ramp mode is selected. In this mode the user can use the Run1 (Control Supervisor Attribute 3) and Run2 (Control Supervisor Attribute 4) commands to alternate between 2 pre downloaded single ramp starting profiles. The use of the Run1 attribute, starts the motor using the RampTime1 and Initial Voltage1 values. The use of the Run2 attribute, starts the motor based on the RampTime2 and InitialVoltage2 values. All other Output Configuration Attributes are common to both ramps, except for potential vendor specific settings of the StopMode (see Stopping Behavior section).

### **5-33.5.1.2 Current Limit Starting**

Current Limit based starting (with no voltage ramp) is selected by setting the StartMode (Attribute 4) to a value of “2”. In this case the output voltage is increased until the motor current equals the CurrentLimitSet (Attribute 16). The voltage is then maintained at a level that does not allow the current Limit to be exceeded, until either full speed is reached (and motor current drops below the CurrentLimitSet) or until the overload is tripped. The Current Limit may only be exceeded during operation of the Kick Start function. CurrentLimitSet(Attribute 16) is entered as a percent of the Motor Data Object Instance attribute RatedCurrent (Attribute 6) value.

### **5-33.5.1.3 Voltage Ramp With Current Limited Starting**

Voltage Ramp with Current Limit based starting is selected by setting StartMode (Attribute 4) to a value of “3”. In this case the output voltage increases according to the selected voltage ramp until the CurrentLimitSet (Attribute 16) is reached. At this point, the output voltage stops increasing and current limit based starting continues until either full speed is reached (and motor current drops below the CurrentLimitSet) or until the overload is tripped.

CurrentLimitSet (Attribute 16) is entered as a percent of the Motor Data Object Instance attribute RatedCurrent (Attribute 6) value.

### **5-33.5.2 Stopping Behavior**

Several optional Stop Modes; Coasting, Ramp Down, DC Braking, or Vendor Specific may be selected via StopMode (Attribute 5). The Stop Mode chosen is independent of the devices Start Mode setting.

#### **5-33.5.2.1 Coasting to Stop**

In this mode, the soft starter sets the output to zero volts, and the motor “coasts” to stop.

#### **5-33.5.2.2 Ramp to Stop**

In this mode, the output voltage is decreased at a fixed rate from line voltage to zero volts over the user defined time period Decel Time (Attribute 16).

#### **5-33.5.2.3 DC Brake to Stop**

For fast motor stopping, a DC braking current is applied to the motor for the duration of the DecelTime (Attribute 15). The amount of DC braking current applied to the motor is specified in BrakingCurrentSet (Attribute 17). BrakingCurrentSet (Attribute 17) is entered as a percent of the Motor Data Object Instance attribute RatedCurrent (Attribute 6) value.

#### **5-33.5.2.4 Vendor Specific Stopping**

In this mode, the soft starter stops the motor by setting the output voltage according to a profile unique to a particular vendor.

### **5-33.5.3 Other Attribute Behaviors**

#### **5-33.5.3.1 Rotation (Attribute 11)**

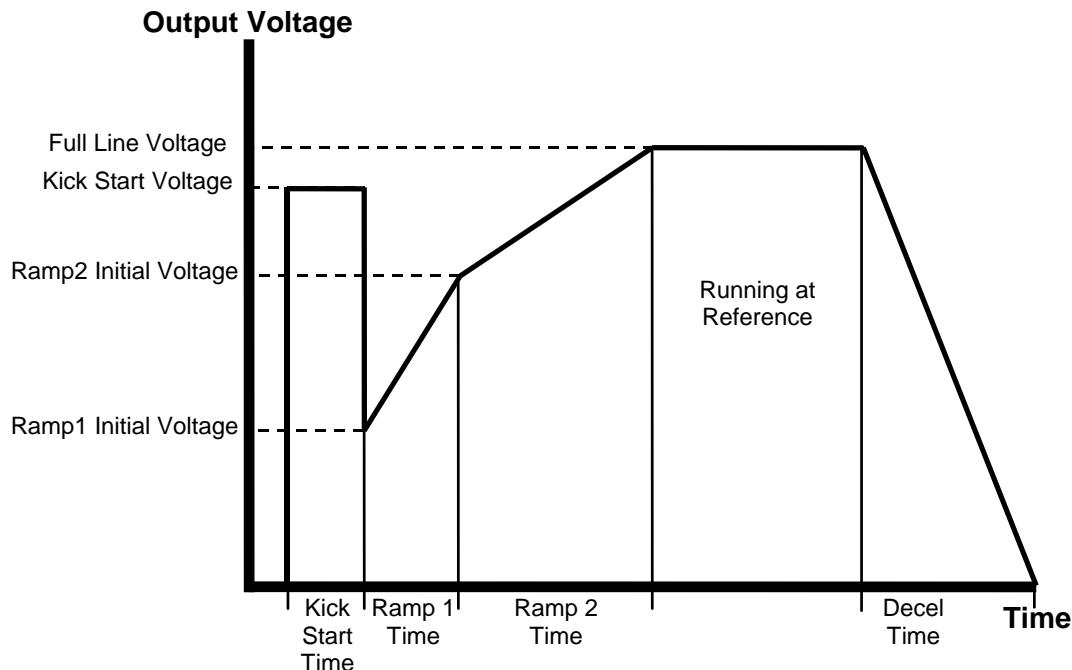
Sets the main voltage phase rotation sequence for Phases A, B & C.

#### **5-33.5.3.2 EnergySaver (Attribute 14)**

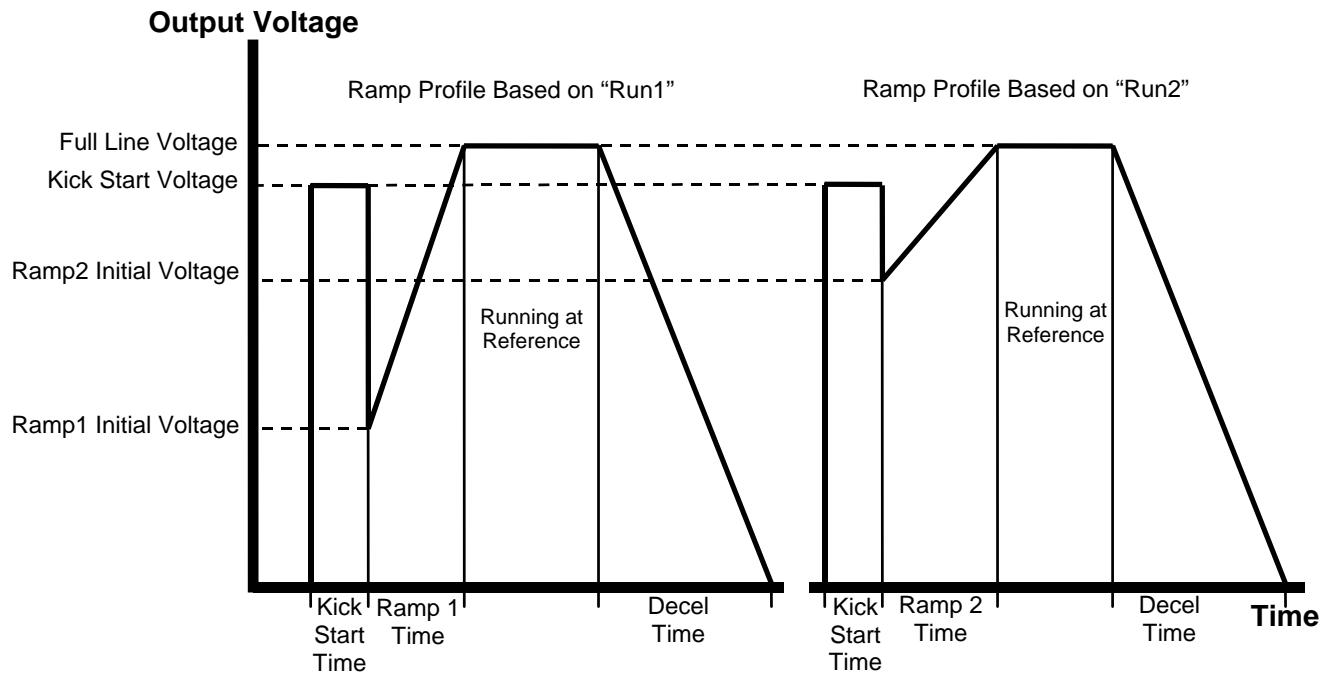
Sets whether the energy saver function is active when the motor is operating at reference. The energy saving function works as follows: When a motor is operating under a loaded condition and then the load is reduced, the current the motor draws will decrease. The energy saver function lowers the voltage to a point where the current starts to increase again. The output voltage is maintained at that level until the load increases, in which case the soft starter will increase the output voltage back up to the line voltage level.

### 5-33.5.4 Soft Starter Output Voltage Behavior Description

**Figure 5-33.4 Example: Dual Contiguous Ramp Soft Start With Ramp Down Stop Mode**



**Figure 5-33.5 Example: Dual Independent Ramp Soft Start With Ramp Down Stop Mode**



## 5-34 Selection Object

**Class Code: 2E<sub>hex</sub>**

The Selection Object manages the selection and distribution of data between objects.

Revision History

**Table 5-34.1 Selection Object Revision History**

Revision	Reason for object definition update
01	Initial Release
02	Modified for use as general data flow manager and added NV column to attribute table

### 5-34.1 Class Attributes

**Table 5-34.2 Selection Object Class Attributes**

Number	Need in Implement	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
1	Conditional *	Get	NV	Revision	UINT	Revision of this object	The current value assigned to this attribute is two (2).
2 thru 7	These class attributes are optional and are described in Chapter 4 of this specification.						
8	Reserved for CIP						
9	Optional	Get	NV	Max number of instances	UINT	Indicates maximum number of instances supported.	Range 0 – 65535.

\*This attribute is REQUIRED if the application supports revision (2) of this object, otherwise the attribute is OPTIONAL.

### 5-34.2 Class Services

**Table 5-34.3 Selection Object Class Services**

Service Code	Need In Implementation	Service Name	Service Description
08hex	Optional	Create	Used to instantiate a next instance of the class.
09hex	Optional	Delete	Used to delete all instances of class.
06hex	Optional	Start	Used to start all instances of class
07hex	Optional	Stop	Used to stop all instances of class.
05hex	Optional	Reset	Used to reset all instances of class. Clear table entries.
0Ehex	Conditional*	Get_Attribute_Single	Used to read a class attribute value.

\*This Service is REQUIRED if any Class attributes are supported.

### 5-34.3 Instance Attributes

**Table 5-34.4 Selection Object Instance Attributes**

Attr ID	Need in Implem	Access Rule	NV	Attribute Name	Data Type	Attribute Description	Semantics of Values
1	Required	Get	V	State	USINT	State of the object.	See attribute description.
2	Required	Get	NV	Max_destinations	UINT	Indicates maximum number of destination	Range 0-65535. 0 = No destination

Selection Object, Class Code: 2E<sub>Hex</sub>

Attr ID	Need in Implem	Access Rule	NV	Attribute Name	Data Type	Attribute Description	Semantics of Values
						indexes supported.	specified.
3	Conditional	Get	NV	Number_of_destinations	UINT	Number of entries of destination_list.	Range 0-65535.
4	Conditional	Set	NV	Destination_list	Array of STRUCT of	List of destination paths	Path length, Path. See Appendix C for encoding.
					{USINT	Pathlength	
					EPATH}	Destination Path	
5	Required	Get	NV	Max_Sources	UINT	Indicates maximum number of source indexes supported.	Range 0-65535. 0 = No Source specified.
6	Required	Get	NV	Number_of_sources	UINT	Number of entries of source_list.	Range 0-65535.
7	Conditional	Set	NV	Source_list	Array of UINT	Array of consuming I/O Connection Instance IDs.	Connection Instance IDs.
8	Conditional	Get, Set*	NV	Source_used	UINT	Index into source_list array indicating which source is currently being used.	Range 0-number of sources, where one (1) is first entry in source_list.
9	Optional	Get	V	Source_Alarm	Array of BOOL	Indicates sources that are not available.	0 = Available, 1 = Not available, Default values are all 0.
10	Optional	Get/Set	NV	Algorithm_Type	USINT	The selection algorithm type.	If not supported, algorithm_type =0.
11	Conditional	Get/Set	NV	Detection_Count	USINT	Required when algorithm type is supported.	Range 2-255. Default value is 4.
12	Conditional	Get/Set	NV	Selection_Period	UINT	Required when algorithm type is supported.	Range 1-65535 (ms).
13	Conditional	Set	NV	object_source_list	ARRAY of STRUCT of	List of source paths	Path length,
					{ USINT	Pathlength,	
					EPATH }	Source Path	
14	Optional	Set	NV	destination_used	UINT	Index into destination_list array indicating which destination is currently being used	Range 0 to number_of_destinations, where (1) is first entry in destination_list. See Behavior
15	Conditional	Set*	NV	input_data_value	determined by source data type	a copy of the currently selected source attribute	See Device Profiles where used for Restrictions on Data Type
16	Conditional	Get	V	output_data_value	determined by destination data type	a copy of the currently selected destination attribute	See Device Profiles where used for Restrictions on Data Type

\* An application may restrict access to Get Only, generally based on the Algorithm types supported.

## 5-34.4 Semantics

### 5-34.4.1 State

Defines the current state of the Selection Object instance. The table below defines the possible states and assigns a value used to indicate that state.

**Table 5-34.5 Values assigned to the state attribute**

Value	State Name	Description
00	Non-existent	The Selection Object does not exist.
01	Idle	The Selection Object does not distribute messages and the internal selection algorithm is not active.
02	Running	The Selection Object distributes messages and internal selection algorithm is running.

### 5-34.4.2 max\_destinations

Maximum number of destination paths managed by the Selection Object.

### 5-34.4.3 number\_of\_destinations

Number of destination paths managed by the Selection Object. Default value is zero(0). This attribute is Required unless max\_destinations = 0.

### 5-34.4.4 destination\_list

This attribute is an array of paths to objects which receive messages from the Selection Object. See Volume I, Appendix C. All members in the list must be unique. This attribute is Required unless max\_destinations = 0.

### 5-34.4.5 max\_sources

Maximum number of source paths managed by the Selection Object.

### 5-34.4.6 number\_of\_sources

Number of source paths managed by the Selection Object. Default value is zero(0). The number\_of\_sources and number\_of\_destinations do NOT have to be equal. This attribute is Required unless max\_sources = 0.

### 5-34.4.7 source\_list

This attribute is an array of Consuming Connection Instance IDs which send messages to the Selection Object. All members in the list shall be unique. This attribute is Required unless (a) max\_sources = 0, or (b) object\_source\_list is supported.

### 5-34.4.8 source\_used

This attribute is used to indicate the source of a message which is selected by the Selection Object. The value of this attribute indicates the index into source\_list currently in use. A value of zero (0) indicates no source is currently being used.

**5-34.4.9 source\_alarm**

This attribute is used to indicate which sources are available. Size of the array is equal to the value of the max\_source attribute.

**5-34.4.10 algorithm\_type**

Defines the algorithm types. If this attribute is supported and any value from 01 through 03 is supported, then support of all values, 00 through 03, is required. If values greater than 03 are supported, then support of the values 00 through 03 is optional. If this attribute is not supported, then all messages pass through. See table below.

**Table 5-35.6 Values assigned to the algorithm\_type attribute**

<b>Value</b>	<b>State Name</b>	<b>Description</b>
00	Pass Through	All messages pass through.
01	Hot Backup*	Hot Backup is used as a selection algorithm.
02	First Arrival*	First Arrival is used as a selection algorithm.
03	Last Arrival*	Last Arrival is used as a selection algorithm.
04	Programmable Data Flow	Programmable Data Flow is used as a selection algorithm.
05-63	Reserved	Reserved for future use (Open).
64-C7	Vendor Specific	
C8 -FF	Reserved	Reserved for future use.

\* If supported, then all values 00 through 03 shall be supported.

**5-34.4.11 detection\_count**

The parameter used by algorithms, which require a counter.

**5-34.4.12 selection\_period**

The parameter used by algorithms, which require a time interval.

**5-34.4.13 Object Source List**

This attribute is an array of paths from which data is sourced. This attribute is Required unless (a) max\_sources = 0, or (b) source\_list is supported.

**5-34.4.14 Destination Used**

This attribute is used to indicate the destination of data. The value of this attribute indicates an index into destination\_list currently in use. A value of zero (0) indicates no destination is currently being used.

**5-34.4.15 Input Data Value**

This attribute contains the value retrieved from the Source. This attribute is Required only when max\_sources = 0

### 5-34.4.16 Output Data Value

This attribute contains the value being delivered to the Destination. This attribute is Required only when max\_destinations = 0

## 5-34.5 Instance Services

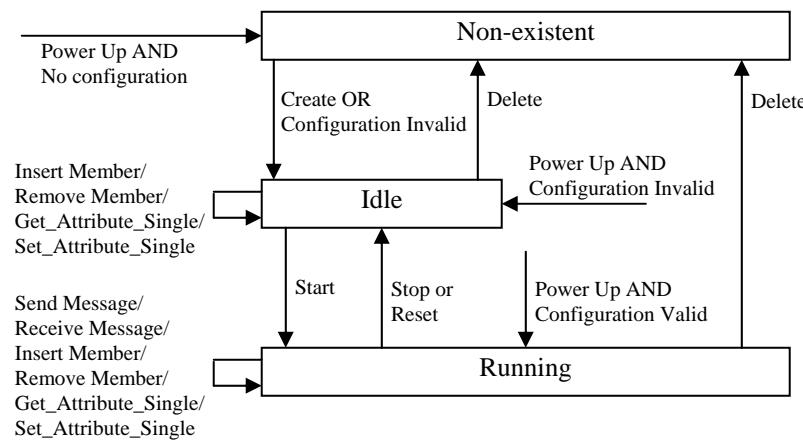
**Table 5-35.7 Selection Object Instance Services**

Service Code	Need In Implementation	Service Name	Service Description
0Ehex	Required	Get_Attribute_Single	Used to read a Selection Object attribute value.
10hex	Optional	Set_Attribute_Single	Used to modify a Selection Object attribute value.
06hex	Optional	Start	Used to start a Selection Object.
07hex	Optional	Stop	Used to stop a Selection Object.
05hex	Optional	Reset	Used to reset a Selection Object. Clear paths.
09hex	Optional	Delete	Used to delete a Selection Object.
18 hex	Optional	Get_Member	Used to get values of a member in an array.
19 hex	Optional	Set_Member	Used to set values to a member in an array.
1A hex	Optional	Insert_Member	Used to insert or add a member to an array.
1B hex	Optional	Remove_Member	Used to remove a member from an array.

## 5-34.6 Instance Behavior

The behavior of the Selection Object, while in the RUNNING state, is dependent upon the algorithm\_type attribute.

**Figure 5-34.8 The Selection Object State Transition Diagram**



Selection Object, Class Code: 2E<sub>Hex</sub>

Table 5-34.9 State Event Matrix for Selection Object

Event	Non-Existent	Idle	Running
Create	Class instantiates a Selection Object. Set default values. Transition to <b>Idle</b> .	Not applicable.	Not applicable.
Power Up & Configuration Valid	Transition to <b>Running</b> .	Not applicable.	Not applicable.
Delete	Error: Object does not exist.	Release all associated resources. Transition to <b>Non-existent</b> .	Release all associated resources. Transition to <b>Non-existent</b> .
Start	Error: Object does not exist.	If attributes are valid, Transition to <b>Running</b> .	Error: The object cannot perform the requested service in its current mode/state.
Stop	Error: Object does not exist.	Error: The object cannot perform the requested service in its current mode/state.	Transition to <b>Idle</b> .
Reset	Error: Object does not exist.	Error: Set default values.	Discard all received messages. Set default values. Transition to <b>Idle</b> .
Receive_Message	Not applicable	Discard the message.	When a message is received from a Connection Object, check source_list attribute and update source_used (if applicable). See algorithm_type for details. If fault detected, set appropriate source index bit in source_alarm attribute.
Remove_Member	Error: Object does not exist.	Remove one member from source_list or destination_list attribute. Subtract one (1) from number_of_sources or number_of_destinations attribute value.	Remove one member from source_list or destination_list attribute. Subtract one(1) from number_of_sources or number_of_destinations attribute value.
Insert_Member	Error: Object does not exist.	Insert or add one member to source_list or destination_list attribute. Algorithm type shall determine usage of members in array.	Insert or add one member to destination_list attribute. Add one(1) to number_of_destinations attribute value. A member cannot be added to source_list attribute. In this case, return error.
Get_Attribute_Single Set_Attribute_Single Get_Member Set_Member	Error: Object does not exist.	Validate/service the request based on internal logic and per the Access Rules.	Validate/service the request based on internal logic and per the Access Rules.

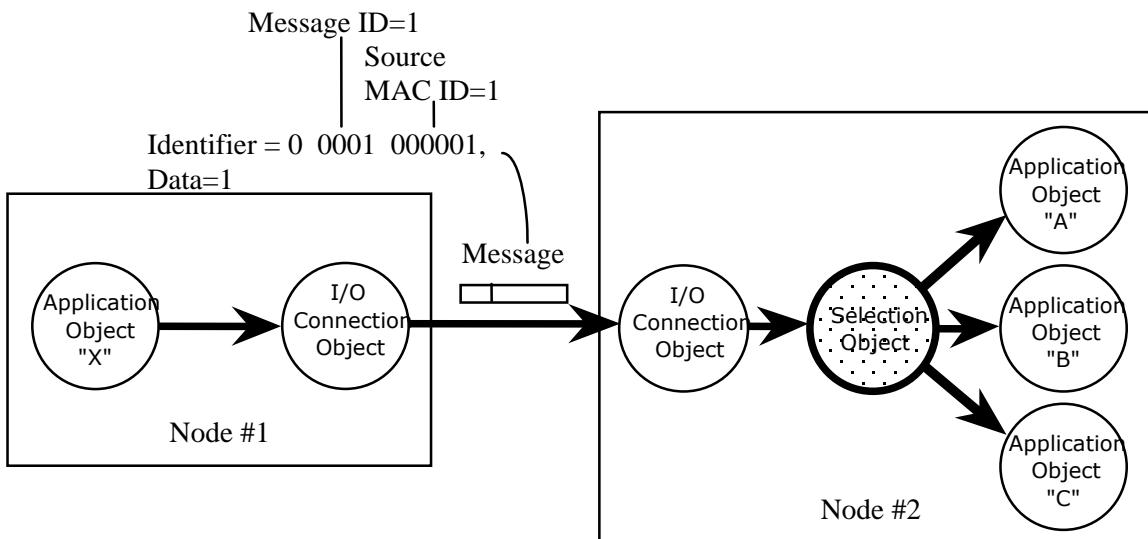
**Table 5-34.10 Selection Object Attribute Access**

Attribute	Non-Existent	Idle	Running
State	Not available	Get only	Get only
max_destinations	Not available	Get only	Get only
number_of_destinations	Not available	Get only	Get only
destination_list	Not available	Get/Set/Insert/Remove	Get/Set/Insert/Remove
max_sources	Not available	Get only	Get only
number_of_sources	Not available	Get only	Get only
source_list	Not available	Get/Set/ Insert /Remove	Get/Set/Remove
source_used	Not available	Get only	Get only
source_alarm	Not available	Get only	Get only
algorithm_type	Not available	Get/Set	Get only
detection_count	Not available	Get/Set	Get/Set
selection_period	Not available	Get/Set	Get/Set

**5-34.6.1 Message Distribution**

The Selection Object manages distribution of messages. This function provides multicast communication among Application Objects. The following diagram illustrates the message flow of this Multicast Communication.

The Selection Object receives a message from a Connection Object and delivers this message to Application Object(s).

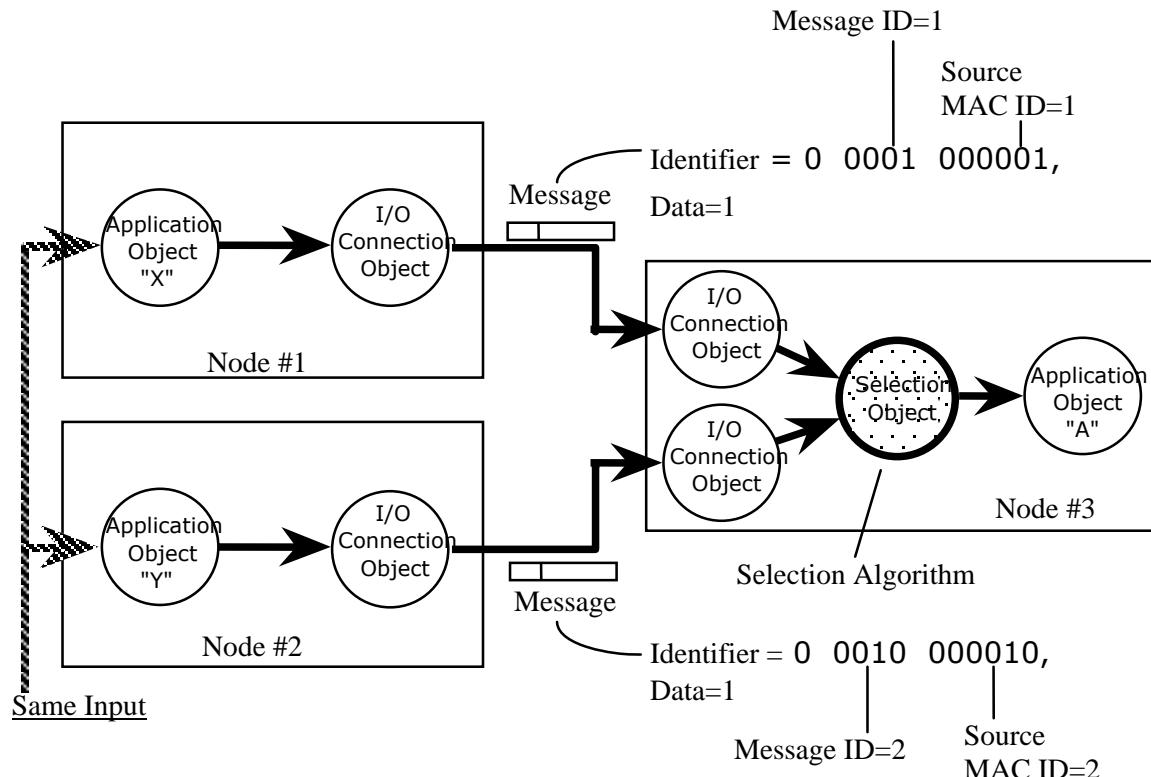
**Figure 5-34.11 I/O Message Distribution**

### 5-34.6.2 Message Selection

The Selection Object may produce one output message from multiple input messages. This function may be used for redundant node applications. The Selection Object selects one message from more than one message based on its internal algorithm.

In high reliability systems, there is a requirement for redundant nodes. The Selection Object receives messages from redundant nodes and selects one message based on an internal selection algorithm.

**Figure 5-34.12 Message Selection Example**



### 5-34.6.3 Message Selection Algorithms

The following algorithms are currently defined for message selection.

- Pass Through
- Hot Backup
- First Arrival
- Last Arrival
- Programmable Data Flow

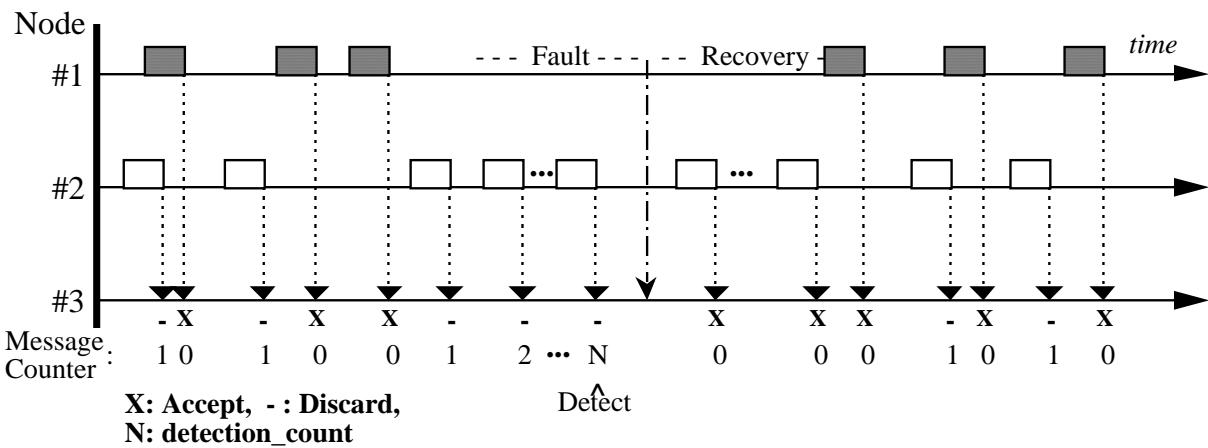
#### 5-34.6.3.1 Pass Through

No algorithm. Forward all arriving messages to destination(s).

### 5-34.6.3.2 Hot Backup

This algorithm is shown in the Figure labeled Hot Backup. Node #1 is higher priority than Node #2. Node #3 normally accepts messages from Node #1. Node failure is detected in the following manner. If the Selection Object detects that Node #1 failed, it accepts messages from Node #2 and forwards them to Application Object(s). Upon the receipt of the message from Node #2, the “Message Counter” within the Selection Object is incremented by one. If the message from Node #1 is received, the “Message Counter” is reset. If the number of the “Message Counter” reaches detection\_count (N), the Selection Object detects that Node #1 failed. If Node #1 recovers and resumes sending messages, the Selection Object accepts messages from Node #1. “N” is the configurable maximum count and set in attribute 12. If “number\_of\_sources”, attribute 7, equals two(2) then the first member in the “source\_list”, attribute 8, has the higher priority.

**Figure 5-34.13 Hot Backup**



### 5-34.6.3.3 First Arrival

This algorithm is shown in the Figure labeled First Arrival. The Selection Object within Node #3 always accepts the first arriving message from either of the two nodes; Node #1 or Node #2. Upon receipt of the first message from an active node, the “Message Counter” within the Selection Object is incremented by one. The “Message Counter” is decremented each time a message is received from the alternate node. If the number within the “Message Counter” reaches “detection\_count”, attribute 12, the Selection Object detects that a node failed. The node to which the positive count is associated determines the node from which the Selection Object forwards the message. “N” is the configurable maximum count contained within attribute 12. The “number\_of\_sources”, attribute 7, equals two(2). See the Figure labeled Behavior of First Arrival in “Running State” for detail behavior.

Figure 5-34.14 First Arrival

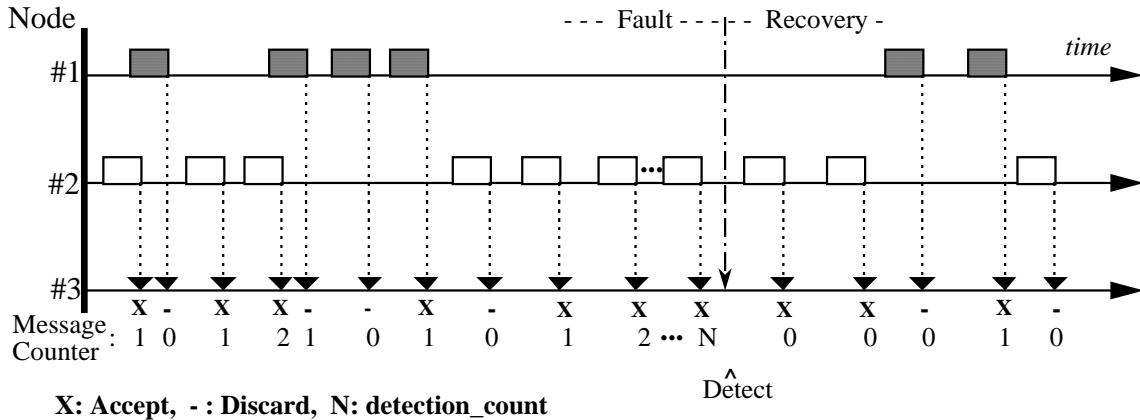
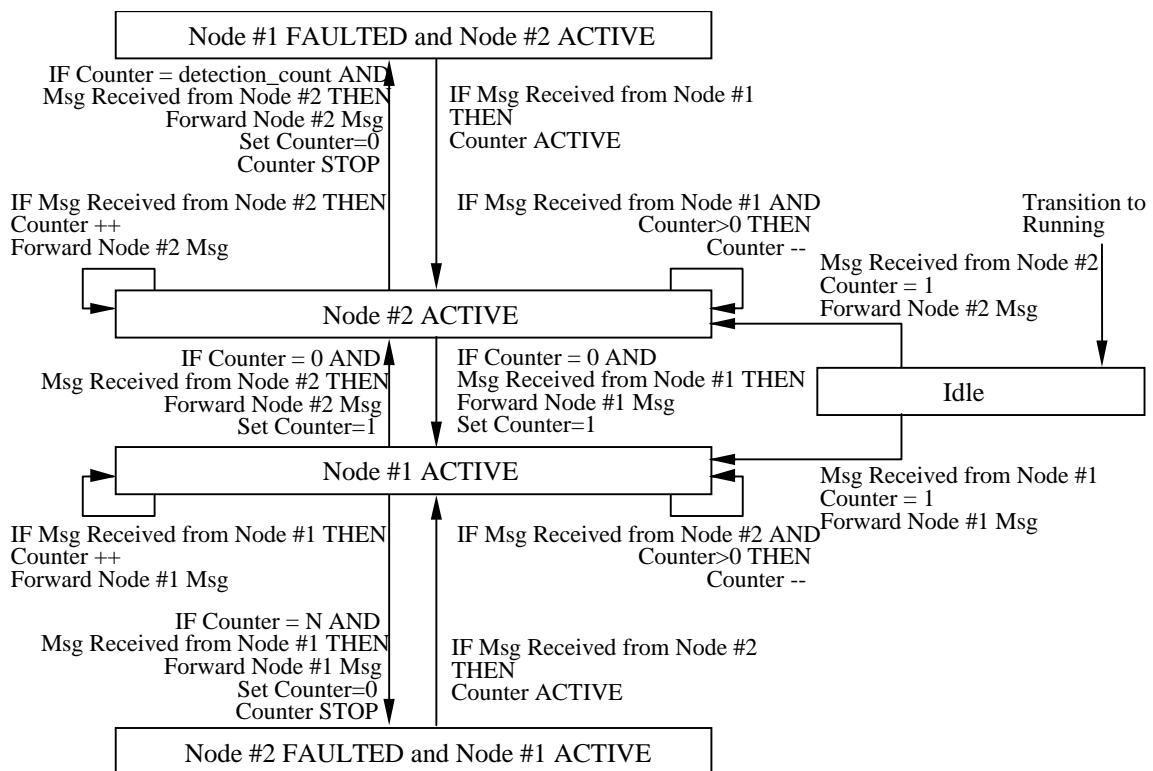


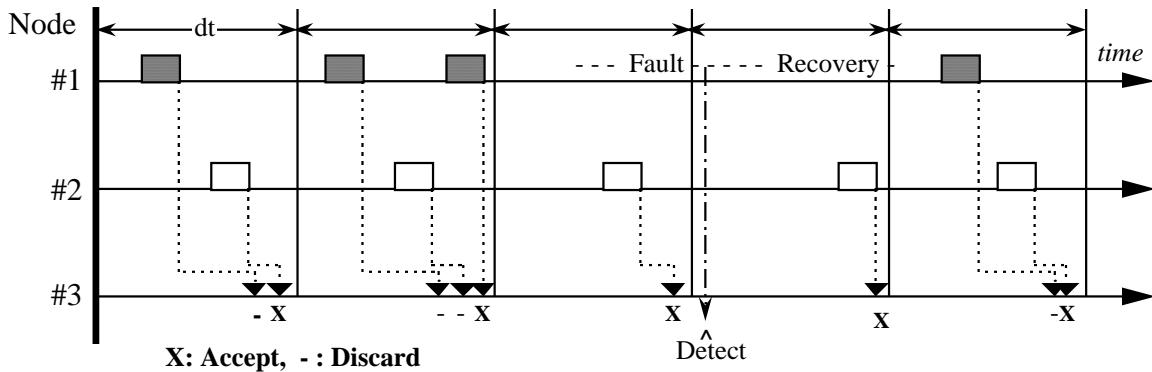
Figure 5-34.15 Behavior of First Arrival in “Running State”



### 5-34.6.3.4 Last Arrival

This algorithm is shown in the Figure labeled Last Arrival. The Selection Object starts a timer when transmits to running. It forwards the last message received when the timer expires, and immediately restarts the timer. If no message is received AND the timer expires, the Selection Object does not forward a message to the Application Object(s). The “selection\_period” which is shown as “dt” in the figure is the value contained in attribute 13. The “number\_of\_sources”, attribute 7, is equal to two(2) in the example.

**Figure 5-34.16 Last Arrival**



### 5-34.6.3.5 Programmable Data Flow

When selected, this algorithm affects the following object behavior:

The Selection object uses the *source\_used* attribute to select a Source EPATH from the *object\_source\_list*. This specifies the source from which the Selection object instance retrieves data values. Similarly, the *destination\_used* attribute is used to select a Destination EPATH from the *destination\_list* in order to specify the destination to which the Selection object instance delivers the retrieved data values. Optionally, (as specified by the application) the Selection object may perform some level of data type conversion. If this capability is specified by a Device Profile or Vendor, the extent to which it is supported shall be specified. The combined functions of Retrieval, Data Type Conversion and Delivery constitute the Application Process for the Selection object while executing the Programmable Data Flow algorithm.

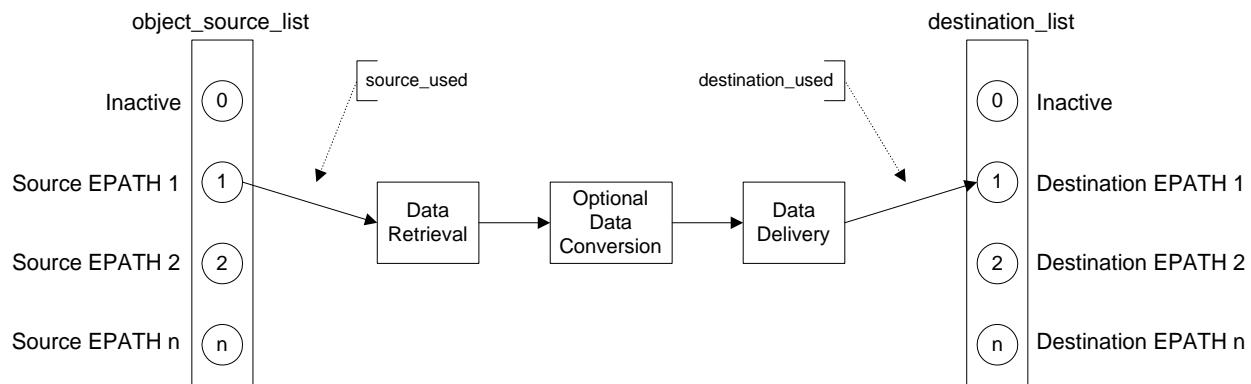
A *source\_used* value of zero (0) halts the Retrieval portion of the Application Process. The retrieval and subsequent setting of the *input\_data\_value* is the only behavior effected by halting the Retrieval Application Process (i.e., *source\_used* = 0). The *input\_data\_value* retains the last value set and is then accessible for manual (or I/O) setting. Similarly, a *destination\_used* value of zero (0) halts the Delivery portion of the Application Process providing for manual (or I/O) setting of a referenced destination and making available for manual retrieval the *output\_data\_value*. With both the Retrieval and Delivery Processes halted, the object's Application Process consists only of the (optional) data conversion process, whereby, the *output\_data\_value* attribute is updated by the object based on the value of the *input\_data\_value* attribute.

**Selection Object, Class Code: 2E<sub>Hex</sub>**

An attempt to set a member of the *source\_list* or *destination\_list* to a source or destination EPATH referencing a data type unsupported by the Selection object returns an Invalid Attribute Value error. Similarly, an attempt to set *source\_used* and/or *destination\_used* to an unsupported value will return an Invalid Attribute Value error; an attempt to set both such that an invalid combination results (i.e., non-matching data types) will return an Object State Conflict.

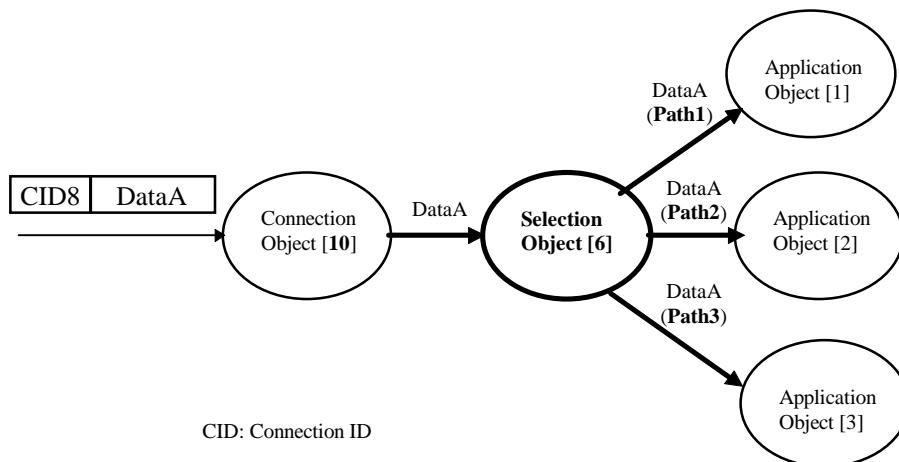
An attempt to set a member of the *object\_source\_list* or *destination\_list* attributes with an EPATH referencing an object class, instance, attribute or structure element that is not known or is not contained in the processing node returns a Path Destination Unknown error.

**Figure 5-34.17 Programmable Data Flow Example**



#### 5-34.6.4 Examples

**Figure 5-34.18 I/O Message Distribution**



**Selection Object, Class Code: 2E<sub>Hex</sub>****Table 5-34.19 I/O Message Distribution Attribute Values**

<b>Attribute Number</b>	<b>Attribute Name</b>	<b>Attribute Value</b>
1	state	02
3	number_of_destinations	3
4	destination_list	{Path1, Path2, Path3}
6	number_of_sources	01
7	source_list	{10}
8	source_used	01
9	source_alarm	{0}
10	algorithm_type	00

**Table 5-34.20 I/O Message Selection Attribute Values**

<b>Attribute Number</b>	<b>Attribute Name</b>	<b>Attribute Value</b>
1	state	02
3	number_of_destinations	1
4	destination_list	{Path1}
6	number_of_sources	02
7	source_list	{10,11}
8	source_used	01
9	source_alarm	{0,0}
10	algorithm_type	02

This page is intentionally left blank

## **5-35 S-Device Supervisor Object**

### **Class Code: 30<sub>Hex</sub>**

This object models the interface, functions and behavior associated with the management of application objects for devices within the “*Hierarchy of Semiconductor Equipment Devices*”. Throughout this CIP Standard, objects belonging to this hierarchy are identified as such by a naming convention that includes a prefix of “S-” in the object class name. This “*Hierarchy of Semiconductor Equipment Devices*” is completely defined in this object definition such that all objects belonging to this hierarchy require the existence of an S-Device Supervisor object to manage its functions and behaviors.

The S-Device Supervisor object centralizes application object state definitions and related status information, exception status indications (alarms and warnings), and defines a behavior model which is assumed by objects identified as belonging to the *Hierarchy of Semiconductor Equipment Devices*. That is, if a reset is requested of the S-Device Supervisor object instance, it will be performed by this object instance as well as all of its associated application objects.

Similarly, the Identity object provides an interface to the S-Device Supervisor object. A reset request to the Identity object (of any type) causes a reset request to the S-Device Supervisor object. Further relationships are specified in the Behavior section below.

Additionally, some device attributes are defined which are required in order to specify device models such that they are compliant with the SEMI S/A Network Standard<sup>1</sup>, from which the *Hierarchy of Semiconductor Equipment Devices* is derived. Objects defined to exist within the *Hierarchy of Semiconductor Equipment Devices* are done so in order to simplify the management and description of object behavior while insuring compliance with the SEMI Standard.

**NOTE:** By association with this object, the Start, Stop, Reset, Abort, Recover and Perform\_Diagnostic Services are inherently supported by all objects within the *Hierarchy of Semiconductor Equipment Devices*. Although, for the associated application object instances, these services are not accessible over the network.

---

<sup>1</sup> Semiconductor Equipment and Materials International, Mountain View CA, Standard E54: *Sensor/Actuator Network Common Device Model*.

## 5-35.1 Class Attributes

Table 5-35.1 S-Device Supervisor Object Class Attributes

Attribute ID	Need in implementation	Access Rule	Name	Data Type	Description of Attribute
1	Required	Get	Revision	UINT	= 02
2 thru 7	These class attributes are optional and are described in Chapter 4 of this specification.				
97-98	Reserved by CIP for Future Use				
99	Conditional *	Get	Subclass	UINT	Identifies a subset of additional class attributes, services and behaviors.

\* If the value of Subclass is 00 which identifies "no subclass", then this attribute is OPTIONAL in implementation, otherwise, this attribute is REQUIRED.

### 5-35.1.1 Subclasses

Each class level subclass defines a unique meaning for an overlapping range of class attribute IDs and/or class service IDs. The range for subclass definitions begins at ID 96 and numbers downward for attributes, and ID 63<sub>hex</sub> and numbers downward for services. The subclass for a given class is identified by the value of its Subclass class attribute. The subclasses for this object are specified at the end of this object specification section.

## 5-35.2 Instance Attributes

CIP reserves Attribute ID 100-199 (64<sub>hex</sub>-C7<sub>hex</sub>) for Vendor Defined Attributes. See Volume I, Chapter 4 for more information on Object Definitions.

Table 5-35.2 S-Device Supervisor Object Instance Attributes

Attr ID	Need in Implementation	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Value
1	Optional	Get	NV	Number of Attributes	USINT	Number of Attributes supported by the object instance	
2	Optional	Get	NV	Attribute List	Array of USINT	List of attributes supported by the object instance	
3	Required	Get	NV	Device Type	SHORT STRING	ASCII Text, Max. 8 Characters	See "Semantics" section
4	Required	Get	NV	SEMI Standard Revision Level	SHORT STRING	Specifies the revision level of the SEMI S/A Network Standard to which the device complies.	For this revision, this attribute must be: "E54-0997"
5	Required	Get	NV	Manufacturer's Name	SHORT STRING	ASCII Text, Max. 20 Characters.	See "Semantics" section
6	Required	Get	NV	Manufacturer's Model Number	SHORT STRING	ASCII Text, Max. 20 Characters, Manufacturer Specified	

S-Device Supervisor Object, Class Code: 30<sub>Hex</sub>

Attr ID	Need in Implementation	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Value
7	Required	Get	NV	Software Revision Level	SHORT STRING	ASCII Text, Max. 6 Characters	See "Semantics" section
8	Required	Get	NV	Hardware Revision Level	SHORT STRING	ASCII Text, Max. 6 Characters	See "Semantics" section
9	Optional	Get	NV	Manufacturer's Serial Number	SHORT STRING	ASCII Text, Max. 30 Characters, Manufacturer Specified	See "Semantics" section
10	Optional	Get	NV	Device Configuration	SHORT STRING	ASCII Text, Max. 50 Characters, Manufacturer Specified. Optional additional information about the device configuration.	
11	Required	Get	V	Device Status	USINT		See "Semantics" section
12	Required	Get	V	Exception Status	BYTE		See "Semantics" section
13 Conditional based on Exception Status Bit 7  Conditional based on Size  Conditional based on Size		Get  Conditional based on Size  Conditional based on Size	V  Conditional based on Size  Conditional based on Size	Exception Detail Alarm	STRUCT of:	A Structure of three Structures containing a bit mapped representation of the alarm detail	
				Common Exception Detail	STRUCT of:		
				Size	USINT	Number of Common Detail Bytes	
				Detail	ARRAY of:		See "Semantics" section
				Detail n	BYTE		See "Semantics" section
				Device Exception Detail	STRUCT of:		
				Size	USINT	Number of Device Detail Bytes	
				Detail	ARRAY of:	See Device Profile	
				Detail n	BYTE	See Device Profile	
				Manufacturer Exception Detail	STRUCT of:		
				Size	USINT	Number of Manufacturer Detail Bytes	
				Detail	ARRAY of:	Manufacturer Specified	
				Detail n	BYTE	Manufacturer Specified	

S-Device Supervisor Object, Class Code: 30<sub>Hex</sub>

Attr ID	Need in Implementation	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Value		
14	Conditional based on Exception Status Bit 7	Get	V	Exception Detail Warning	STRUCT of:	A Structure of three Structures containing a bit mapped representation of the warning detail			
				Common Exception Detail	STRUCT of:				
				Size	USINT	Number of Common Detail Bytes			
	Conditional based on Size			Detail	ARRAY of:		See “Semantics” section		
				Detail n	BYTE		See “Semantics” section		
				Device Exception Detail	STRUCT of:				
	Conditional based on Size			Size	USINT	Number of Device Detail Bytes			
				Detail	ARRAY of:	See Device Profile			
				Detail n	BYTE	See Device Profile			
	Conditional based on Size			Manufacturer Exception Detail	STRUCT of:				
				Size	USINT	Number of Manufacturer Detail Bytes			
				Detail	ARRAY of:	Manufacturer Specified			
	Conditional based on Size			Detail n	BYTE	Manufacturer Specified			
15	Required	Set	NV	Alarm Enable	BOOL		See “Semantics” section		
16	Required	Set	NV	Warning Enable	BOOL		See “Semantics” section		
17	Optional	Set	*	Time	DATE_AND_TIME	The value of the device’s internal realtime clock.	See “Semantics” section		

S-Device Supervisor Object, Class Code: 30<sub>Hex</sub>

Attr ID	Need in Implementation	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Value
18	Optional	Get	NV	Clock Power Cycle Behavior	USINT	Describes the behavior of the device's internal realtime clock (the Time attribute) during a power cycle	0 = [default] clock always resets during power cycle 1 = clock value is stored in non-volatile memory at power down 2 = clock is battery-backed and runs without device power. 3-255 = Reserved
19	Optional	Get	NV	Last Maintenance Date	DATE	The date on which the device was last serviced.	
20	Optional	Get	NV	Next Scheduled Maintenance Date	DATE	The date on which it is recommended that the device next be serviced.	
21	Optional	Get	NV	Scheduled Maintenance Expiration Timer	INT		See "Semantics" section
22	Conditional - Required if Calibration Expiration is supported	Set	NV	Scheduled Maintenance Expiration Warning Enable	BOOL		See "Semantics" section
23	Optional	Get	NV	Run Hours	UDINT	An indication of the number of hours that the device has had power applied. It has a resolution of 1 hour. This value shall be maintained in nonvolatile memory.	
24	Optional	Get	V	Endpoint	BOOL	Indicates that the device has determined that the conditions of its endpoint algorithm are satisfied for a particular recipe step.	0 = Endpoint not obtained 1 = Endpoint obtained See Semantics section
25	Optional	Set	NV	Recipe	UINT	Provides a common mechanism for selecting endpoint algorithms.	See Semantics section
81-96	Defined by subclasses below						
97-98	Reserved by CIP for Future Use						

**S-Device Supervisor Object, Class Code: 30<sub>Hex</sub>**

Attr ID	Need in Implementation	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Value
99	Conditional **	Get	NV	Subclass	UINT	Identifies a subset of additional instance attributes, services and behaviors.	0 = No subclass 1 = Power Generator

\* The nonvolatile behavior of the *Time* attribute is conditional on the value of the *Clock Power Cycle Behavior* attribute.

\*\* If the value of Subclass is 00, which identifies "no subclass" then this attribute is OPTIONAL in implementation, otherwise, this attribute is REQUIRED.

**5-35.2.1 Subclasses**

Each instance level subclass defines a unique meaning for an overlapping range of instance attribute IDs and/or instance service IDs. The range for subclass definitions begins at ID 96 and numbers downward for attributes, and ID 63<sub>hex</sub> and numbers downward for services. The subclass for a given instance is identified by the value of its Subclass instance attribute. The subclasses for this object are specified at the end of this object specification section.

**5-35.3 Semantics****5-35.3.1 Device Type**

The Device Type attribute, identifies the Specific Device Model to which the device is modeled within the *Hierarchy of Semiconductor Equipment Devices*. The value of this string is specified in the SEMI standard suite referenced in the introduction section of this object definition and is represented for reference in the applicable device profile where used.

**5-35.3.2 Manufacturer's Name**

The Manufacturer's Name attribute, identifies the manufacturer of the device. It is the responsibility of the manufacturer to insure that this ASCII coded text string is sufficiently long to insure uniqueness among manufacturers.

The Device Manufacturer attribute is not guaranteed, by specification, to be unique. Therefore, it is not a substitute for the corresponding attribute of the Identity Object and should not be used for identification purposes.

**5-35.3.3 Software Revision Level**

This is an ASCII coded text string representing the revision of the software corresponding to the specific device identified by the Identity object and the S-Device Supervisor object.

**5-35.3.4 Hardware Revision Level**

This is an ASCII coded text string representing the revision of the hardware which is identified by the Identity object and the S-Device Supervisor object. The manufacturer of the device must control this revision such that modifications to the device hardware may be tracked.

### 5-35.3.5 Manufacturer's Serial Number

This attribute is a string representation of the vendor's serial number of the device, formatted to fit most manufacturing tracking systems. This is not the same as the Identity Object's serial number which is used to uniquely identify the device in the network environment.

### 5-35.3.6 Device Status

This attribute represents the current state of the device. Its value changes as the state of the device changes. The following values are defined:

**Table 5-35.3 Device Status Attribute Values**

Attribute Value	State
0	Undefined
1	Self Testing
2	Idle
3	Self-Test Exception
4	Executing
5	Abort
6	Critical Fault
7-50	Reserved by CIP
51-99	Device Specific
100-255	Vendor Specific

### 5-35.3.7 Exception Status

A single byte attribute whose value indicates that the status of the alarms and warnings for the device. This indication may be provided in one of two methods: Basic or Expanded.

For the *Basic Method*, bit seven of the Exception Status attribute is set to zero; all exceptions are reported exclusively through communication of this Exception Status attribute. The format of bits zero through six in this mode is device specific; the format may be further specified in an appropriate device profile specification; if it is not specified, then the format of bits zero through six is equivalent to that specified for the expanded method.

For the *Expanded Method*, bit seven of Exception Status attribute is set to one; exceptions are reported through the communication of this Exception Status attribute, formatted as specified in the table below. In addition, the Exception Detail attributes are supported. The Exception Status bits are determined by a logical “OR” of the related Exception Detail bits, as indicated.

**Table 5-35.4 Exception Status Bit Map**

Bit	Exception Status Bit Map, Bit 7 set to 0	Exception Status Bit Map, Bit 7 set to 1
	Function	Function
0	Device Specific definition (See Device Profile)	ALARM/device-common*
1		ALARM/device-specific
2		ALARM/manufacturer-specific
3		reserved -- set to 0
4		WARNING/device-common*
5		WARNING/device-specific
6		WARNING/manufacturer-specific
7	0 == Basic Method	1 == Expanded Method

\* The alarm or warning is not specific to the device type or device type manufacturer.

### 5-35.3.8 Exception Detail Alarm and Exception Detail Warning

The formats of these two attributes are identical. Therefore, they are described together here:

Attributes that relate the detailed status of the alarms or warnings associated with the device. Each attribute is a structure containing three members; these three members respectively relate the detailed status of exceptions that are common (i.e., not device-specific), device-specific but not manufacturer-specific, and manufacturer-specific. The common detail is defined below. The device-specific detail is defined in the appropriate Device Profile. The manufacturer-specific detail is defined by the manufacturer. A SIZE value of zero indicates that no detail is defined for the associated exception detail structure.

Each of the three structure members is defined as a structure containing an ordered list (i.e., array) of bytes of length SIZE, and an unsigned integer whose value is SIZE. Each of the bytes in each array has a specific mapping. This mapping is formatted as 8 bits representing 8 independent conditions, whereas a value of 1 indicates that the condition is set (or present), and a value of 0 indicates that the condition is cleared (or not present). Note that if a device does not support an exception detail, the corresponding bit is never set. The bitmaps for alarms and warnings in the corresponding attributes are structured in parallel so that a condition may have either alarm or warning set depending on severity. If a condition inherently cannot be both alarm and warning, then the parallel bit position corresponding to the other state will remain "0."

The existence of an exception detail variable structure is dependent on the value of the Exception Status Attribute; the existence of an exception detail variable structure is only required if bit seven of the Exception Status attribute is set to 1 (indicating Expanded method reporting) and the bit (among bits zero through six) of the Exception Status attribute corresponding to the particular exception type is also set to 1.

### 5-35.3.9 Common Exception Detail

This structure relates exception conditions (i.e., alarms or warnings) which are common to all devices within the Hierarchy of Semiconductor Equipment Devices. The Detail element of the structure is an ordered list (i.e., array) of bytes of length [SIZE] which is the value of the structure element Size. For each byte in the Detail field, all bits which are not identified are reserved for future standardization.

The first byte in this attribute is CommonExceptionDetail[0]. Additional exception details, if provided, are named CommonExceptionDetail[1], . . . CommonExceptionDetail[SIZE]. The specific exception associated with each of the bitmaps is given in the table below. The SIZE for this revision is two (2). The criteria details for each exception condition are outside the scope of this document.

**Table 5-35.5 Common Exception Detail Attribute Values**

Bit	Common Exception Detail [0]*	Common Exception Detail [1]*
0	Internal diagnostic exception	power supply overcurrent
1	Microprocessor exception	reserved power supply
2	EPROM exception	power supply output voltage
3	EEPROM exception	power supply input voltage
4	RAM exception	scheduled maintenance due
5	Reserved by CIP	notify manufacturer
6	Internal real-time exception	reset exception
7	Reserved by CIP	reserved by CIP

\* Note that if a device does not support an exception detail, the corresponding bit is never set

### 5-35.3.10 Device Exception Detail

This structure, similar in form to Common Exception Detail, relates exception conditions, which are specific to individual devices on the network and are defined in their respective device profiles. The Detail element of the structure is an ordered list (i.e., array) of bytes of length [SIZE], which is the value of the structure element size. For a detailed description of this attribute, consult the appropriate specific device profile.

### 5-35.3.11 Manufacturer Exception Detail

This structure, similar in form to Common Exception Detail, relates exception conditions, which are specific to the manufacturers of individual devices on the network and are defined by them in their product documentation. The Detail element of the structure is an ordered list (i.e., array) of bytes of length [SIZE], which is the value of the structure element Size. For a detailed description of this attribute, consult the appropriate specific device manufacturer documentation.

**Table 5-35.6 Exception Detail Format Summary**

Data Component	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Common Exception Detail Size	0	0	0	0	0	0	1	0
Common Exception Detail 0	Reserved	Real-time Fault	Reserved	Data Memory	Non-Volatile Memory	Code Memory	Micro-processor	Diagnostic
Common Exception Detail 1	Reserved	Reset Exception	Notify Vendor	Scheduled Maint. Due	PS Input Voltage	PS Output Voltage	Reserved	PS Over Current
Device Exception Detail Size	x	x	x	x	x	x	x	x
Device Exception Detail 0	—	—	—	—	—	—	—	—
Device Exception Detail n	—	—	—	—	—	—	—	—
Manufacturer Exception Detail Size	x	x	x	x	x	x	x	x
Manufacturer Exception Detail 0	—	—	—	—	—	—	—	—
Manufacturer Exception Detail n	—	—	—	—	—	—	—	—

### 5-35.3.12 Alarm Enable and Warning Enable

These Boolean attributes are used to enable (1) or disable (0) the S-Device Supervisor object's process of setting Exception bits. When disabled, corresponding bits are never set; and, if they were set, disabling clears them. Also, alarm and warning states are not retained; when enabled, bits will be set only if the corresponding condition is true.

The default state for these Enable attributes is enabled (1).

### **5-35.3.13 Time**

This optional attribute represents the value of the time and date as maintained by the device's realtime clock with a resolution of one millisecond.

The default value for the Time attribute is zero (0), corresponding to 12:00AM, January 1, 1972, as specified by Appendix C.

### **5-35.3.14 Scheduled Maintenance Expiration Timer**

This attribute, with a resolution of one hour, is used to cause a warning which indicates that a device calibration is due. A S-Device Supervisor timer decrements this attribute once per hour while power is applied. When the attribute is no longer positive and the Scheduled Maintenance Expiration Warning Enable attribute is set to enabled, a Scheduled Maintenance Expiration Warning condition is generated. This causes the Scheduled Maintenance Due Warning bit to be set.

The attribute will not wrap; when the attribute reaches its most negative value, it no longer decrements. The attribute will continue to decrement irrespective of the state of the Scheduled Maintenance Expiration Warning Enable attribute. The value shall be maintained in nonvolatile memory.

### **5-35.3.15 Scheduled Maintenance Expiration Warning Enable**

This Boolean attribute is used to enable (1) or disable (0) the S-Device Supervisor object's process of setting the Scheduled Maintenance Due Exception bit. When disabled, the corresponding bit is never set; and, if it was set, disabling clears it. When enabled, the bit will be set only if the corresponding condition is true.

The default state for this Enable attribute is enabled (1).

### **5-35.3.16 Endpoint**

This attribute is used to indicate that the device has determined that the conditions of its endpoint algorithm are satisfied for a particular recipe step. The vendor must specify the endpoint algorithm, including the configuration mechanisms and parameters.

### **5-35.3.17 Recipe**

This attribute provides a common mechanism for selecting endpoint algorithms. This index pointer is used to specify which endpoint algorithm is currently active. The particular algorithm with its associated configuration, limits and other parameters are manufacturer specific.

## 5-35.4 Common Services

Table 5-35.7 S-Device Supervisor Object Common Services

Service Code	Need in Implementation		Service Name	Description of Service
	Class	Instance		
0E <sub>hex</sub>	Conditional	Required	Get_Attributes_Single	Returns the contents of the specified attribute.
10 <sub>hex</sub>	n/a	Required	Set_Attributes_Single	Modifies an attribute value.
05 <sub>hex</sub>	n/a	Required	Reset	Resets the device to the <b>Self-Testing</b> state.
06 <sub>hex</sub>	n/a	Required	Start	Starts the device execution by moving the device to the <b>Executing</b> state. Equivalent to SEMI S/A Network Execute Service
07 <sub>hex</sub>	n/a	Optional	Stop	Moves the device to the <b>Idle</b> state

See Appendix A for definition of these services

## 5-35.5 Object-specific Services

Table 5-35.8 S-Device Supervisor Object Object-specific Services

Service Code	Need in Implementation		Service Name	Description of Service
	Class	Instance		
4B <sub>hex</sub>	n/a	Required	Abort	Moves the device to the <b>Abort</b> state
4C <sub>hex</sub>	n/a	Required	Recover	Moves the device out of the <b>Abort</b> state
4E <sub>hex</sub>	n/a	Required	Perform_Diagnostics	Causes the device to perform a set of diagnostic routines

Table 5-35.9 DS Object Service Parameter Dictionary

Parameter	Form	Description of Service
TestID	USINT	Type and possibly detail of diagnostic test to be performed

### 5-35.5.1 Abort

Used to transition the device application objects to the aborted state. This service request may be (and generally will be) originated internally, from application objects.

### 5-35.5.2 Recover

Used to transition the device application objects, out of the abort state, to the idle state. This service request may be originated internally, from application objects.

### 5-35.5.3 Perform\_Diagnostics

Used to instruct the S-Device Supervisor object to perform a diagnostic test. A diagnostic test is either of type *common* or *device-dependent*. *Common* diagnostic tests could include: RAM, EPROM, non-volatile memory, and communications. The structure of *common* type diagnostic tests are implementation-specific. All detail of *device-dependent* diagnostics is outside the scope of this document.

### 5-35.5.4 TestID Parameter

The following values are defined for the TestID parameter for the Perform\_Diagnostics Service Request:

**Table 5-35. 10 TestID Parameter Attribute Value**

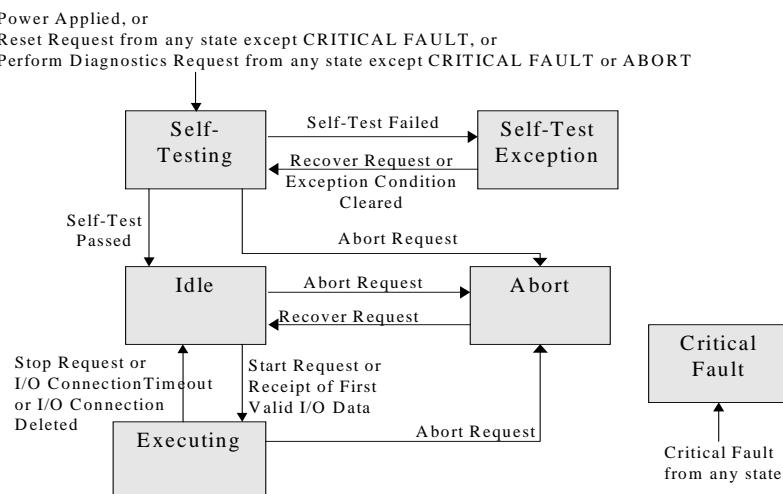
Attribute Value	State
0	Standard
1-63	Reserved
64-127	Device Specific (defined in Device Profile)
128-255	Manufacturer Specific (defined by manufacturer)

Type “Standard” is specified if there is only one type of diagnostic defined or if there are more than one including a type standard. Additional diagnostic types may be defined in the device profile or by the manufacturer.

## 5-35.6 Behavior

### 5-35.6.1 S-Device Supervisor Object States

**Figure 5-35.11 DS Object Instance Behavior**



**Table 5-35.12 DS Instance Behavior State Description**

<b>State</b>	<b>Description</b>
EXECUTING	Device is executing, e.g., it is performing its device specific function. A detailed description of this state is outside the scope of this document. This state may be further specified in an appropriate device profile specification.
SELF TESTING	Object instance exists and has been initialized; all attributes have appropriate initial values (as indicated herein and in applicable device profile specification). Exception Status bits have been reset. Device is performing device-specific and device type specific test to determine if it is qualified to execute its application process.
SELF TEST EXCEPTION	Object has detected an exception condition during self-testing. The details of the exception are stored in the appropriate attribute values of the S-Device Supervisor object.
IDLE	Object and device have been initialized and successfully completed self-testing. Further, the device is not executing the operational components of its device specific functions.
ABORT	Object instance is in an aborted state; a detailed description of this state is outside the scope of this specification.
CRITICAL FAULT	The object (and device) are in a fault state from which there is no recovery. object services cannot be processed. The conditions required for exit from a critical fault is outside the scope of this specification.

The S-Device Supervisor Status attribute value indicates the state of the S-Device Supervisor object. It is updated on appropriate state transitions within the S-Device Supervisor object. Attribute values 1 through 6 represent valid states. A value of zero indicates that the S-Device Supervisor state is unknown; conditions under which a zero value may occur are outside the scope of this document.

## 5-35.6.2 State Event Matrix

Table 5-35.13 State Event Matrix for S-Device Supervisor Object

Event	State					
	Idle	Self-Testing	Self-Test Exception	Executing	Abort (Recoverable Fault)	Critical Fault
Power Applied	—	Default Entry Point: Device performs its Self-Test Application Process	—	—	—	Transition to SELF-TESTING
Self-Test Passed	Not Applicable	Transition to IDLE	Not Applicable	Not Applicable	Not Applicable	Not Applicable
Self-Test Failed	Not Applicable	Set appropriate Exception Status Bits and Transition to SELF-TEST EXCEPTION	Not Applicable	Not Applicable	Not Applicable	Not Applicable
Exception Condition Cleared	Not Applicable	Not Applicable	Set appropriate Exception Status Bits and Transition to SELF-TESTING	Not Applicable	Not Applicable	Not Applicable
Critical Fault	Transition to CRITICAL FAULT	Transition to CRITICAL FAULT	Transition to CRITICAL FAULT	Transition to CRITICAL FAULT	Transition to CRITICAL FAULT	Ignore Event
Reset Request	Transition to SELF-TESTING	Restart SELF-TESTING	Transition to SELF-TESTING	Transition to SELF-TESTING	Transition to SELF-TESTING	Ignore Event
Start Request	Transition to EXECUTING	Error OSC*	Error OSC*	Error AIRS*	Error OSC*	Ignore Event
Stop Request	Error AIRS*	Error OSC*	Error OSC*	Transition to IDLE	Error OSC*	Ignore Event
Abort Request	Transition to ABORT	Transition to ABORT	Error OSC*	Transition to ABORT	Error AIRS*	Ignore Event
Recover Request	Error OSC*	Restart SELF-TESTING	Transition to SELF-TESTING	Error OSC*	Transition to IDLE	Ignore Event
Perform Diagnostics Request	Transition to SELF-TESTING	Restart SELF-TESTING	Transition to SELF-TESTING	Transition to SELF-TESTING	Perform all device diagnostics test.	Ignore Event
I/O Connection Timeout	Ignore Event	Ignore Event	Ignore Event	Transition to IDLE	Ignore Event	Ignore Event
Receipt of First Valid I/O Data	Transition to EXECUTING	Ignore Event	Ignore Event	Normal Response	Ignore Event	Ignore Event
I/O Connection Deleted	Ignore Event	Ignore Event	Ignore Event	Transition to IDLE	Ignore Event	Ignore Event

\* Error AIRS = Error Response “Already in Requested Mode/State” (Code 0B<sub>hex</sub>)Error OSC = Error Response “Object State Conflict” (Code 0C<sub>hex</sub>)

Any S-Device Supervisor instance service may be requested internally by the device as specified by the manufacturer. Generally, these requests will be generated as the result of an event such as the activation of a button or external contact closure.

### 5-35.6.3 S-Device Supervisor and Identity Object Interface

The following two tables describe the effects that the Identity object and the S-Device Supervisor object have on each other based on events. This event mapping defines the interface between these two objects.

**Table 5-35.14 Effect of Identity Object Event**

Identity Object Event	S-Device Supervisor Object Affected Event
Power Applied	Power Applied
Failed Tests	Self-Test Failed
Passed Tests	Self-Test Passed
Major Unrecoverable Fault	Critical Fault
Minor Fault	No effect
Fault Corrected	Exception Condition Cleared
Reset	Reset Request *

\* The S-Device Supervisor object service requests are generated by the device in response to these events.

**Table 5-35.15 Effect of S-Device Supervisor Object Event**

S-Device Supervisor Object Event	Identity Object Affected Event
Power Applied	Power Applied
Self-Test Passed	Passed Tests
Self-Test Failed	Failed Tests
Exception Condition Cleared	Fault Corrected
Critical Fault	Major Unrecoverable Fault
Perform Diagnostics Request	No effect.*

\* The request to perform diagnostics has no effect on the Identity Object. However, the results of the tests may.

### 5-35.6.4 S-Device Supervisor and Application Object Interface

When processing Stop, Start, Reset, Abort, Recover and Perform\_Diagnostics service requests, the S-Device Supervisor object coordinates all Application Objects within the device, as appropriate, in order that they will exhibit behavior consistent with the S-Device Supervisor object instance state.

This object interfacing is best described in the SEMI standard suite as referenced in the introduction section of the object definition.

## 5-35.7 S-Device Supervisor Instance Subclass 01 — Power Generator

The following specification applies to a subclass of this object for application in Power Generator devices such as those used to create plasma energy.

### 5-35.7.1 Instance Attributes

The following Instance Attributes are specified for this object subclass.

**Table 5-35.16 Power Generator Instance Attributes**

Attr ID	Need in Implementation	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
81	Conditional*	Set	NV	Energy Control Enabled	BOOL	Controls the Energy Control Application Process (See Behavior)	0 = Disabled [default] 1 = Enabled
82	Conditional*	Set	NV	Joules to Deliver	UDINT	Controls the Energy Control Application Process (See Behavior)	Joules [default] = 0
83	Conditional*	Get	V	Joules Remaining	UDINT	Controls the Energy Control Application Process (See Behavior)	Joules [default] = 0
84	Optional**	Set	NV	Active Target ID	UINT	A numerical index referring to which target is being tracked via the Active Target Life Accumulator	[default] = 0
85	Optional**	Set	NV	Active Target Life Counter Enable	BOOL	Enables the Active Target Life counter	[default] = 0
86	Optional**	Set	NV	Active Target Life	DINT	Once set, the value counts down to zero. Upon reaching a zero count, the associated <i>Exception Detail Warning</i> bit is set.	KWH [default] = 0
87	Optional	Get	NV	Input Power On Time	UDINT	Factory reset accumulator	Seconds
88	Optional	Get	NV	Output Power On Time	UDINT	Factory reset accumulator	Seconds
89	Optional	Set	NV	Filament On Time	UDINT	Settable accumulator	Seconds [default] = 0
90	Optional	Get	NV	Total Energy Delivered	UDINT	Factory reset accumulator	KWH
91	Optional	Get	NV	Power On Cycle Counter	UDINT	Factory reset accumulator	Counts
92	Optional	Get	NV	Output On Cycle Counter	UDINT	Factory reset accumulator	Counts
93	Optional	Get	NV	Device Over-temp Event Counter	UDINT	Factory reset accumulator	Counts
94	Optional	Get	NV	Regulation Alarm Event Counter	UDINT	Factory reset accumulator	Counts

**S-Device Supervisor Object, Class Code: 30<sub>Hex</sub>**

Attr ID	Need in Implementation	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
95	Optional	Set	V	Filament Power Enable	BOOL	Causes the filament power to turn on only if the device is in the Executing state	0 = Disables Power [default] 1 = Enables Power
96	Required	Set	V	Output Power Enable	BOOL	Causes the power output to turn on only if the device is in the Executing state	0 = Disables Power [default] 1 = Enables Power

\* Required if Energy Control is supported

\*\* Required if Target Life tracking is supported

### 5-35.7.2 Services

There are no additional services defined for this subclass

### 5-35.7.3 Behavior

#### 5-35.7.3.1 Energy Control Application Process

The Energy Control Application Process makes use of a down counter. When power delivery is started, the Joules Remaining attribute assumes the value of the Joules to Deliver attribute. While power is being delivered, the Energy Accumulator calculates delivered energy and decrements the Joules Remaining attribute accordingly. Upon reaching a value of zero, the power delivery is stopped.

## 5-36 S-Analog Sensor Object

### Class Code: 31hex

The S-Analog Sensor Object models the acquisition of a reading from a physical sensor in a device. Associated with an analog sensor is a reading that has been acquired and corrected with an offset and a gain coefficient, optionally, settable in the object. Additional correction algorithms may be specified by other objects identified in the device profile or as extensions specified by the manufacturer.

This object is a member of the *Hierarchy of Semiconductor Equipment Devices*. As such, its behavior is managed by the S-Device Supervisor Object. See Section 5-35.

### 5-36.1 Class Attributes

**Table 5-36.1 S-Analog Sensor Object Class Attributes**

Attr ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Value
1 thru 7	These class attributes are optional and are described in Chapter 4 of this specification.					
32	Conditional **	Get	Class Level Status Extension	USINT	Indicates whether a combination or multigauges measures above or below its maximum or minimum measurement range	Bit 0: Reading Invalid Bit 1: Overrange Exceeded Bit 2: Underrange Exceeded
94-96	Defined by Subclasses					
97-98	Reserved by CIP for Future Use					
99	Conditional *	Get	Subclass	UINT	Identifies a subset of additional class attributes, services and behaviors.	0 = No subclass 1 = Instance Selector 2 - 65535 = Reserved by CIP

\* If the value of Subclass is 00, which identifies "no subclass", then this attribute is OPTIONAL in implementation, otherwise, this attribute is REQUIRED.

\*\* Required for combination gauges and multigauges only

### 5-36.1.1 Subclasses

Each class level subclass defines a unique meaning for an overlapping range of class attribute IDs and/or class service IDs. The range for subclass definitions begins at ID 96 and numbers downward for attributes, and ID 63<sub>hex</sub> and numbers downward for services. The subclass for a given class is identified by the value of its Subclass class attribute. The subclasses for this object are specified at the end of this object specification section.

### 5-36.2 Instance Attributes

Certain minimal implementations may support any optional "Set" attributes as "Get Only" and still be compliant with this object specification. All required attributes must be supported as specified.

**Table 5-36.2 S-Analog Sensor Object Instance Attributes**

Attr ID	Need in Implementation	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
1	Optional	Get	NV	Number of Attributes	USINT	Number of attributes supported	
2	Optional	Get	NV	Attribute List	ARRAY OF USINT	List of attributes supported by this object instance	
3	Optional	See Semantics	NV	Data Type	USINT	Determines the Data Type of <i>Value</i> and all related attributes as specified in this table.	see Semantics section
4	Optional	See Semantics	NV	Data Units	ENGUNITS	Determines the Units context of <i>Value</i> and all related attributes.	see Semantics section
5	Required	Get	V	Reading Valid	BOOL	Indicates that the <i>Value</i> attribute contains a valid value.	0 = invalid 1 = valid (invalid: e.g., not warmed up yet)
6	Required	Get	V	Value	INT or specified by <i>Data Type</i> if supported	Analog input value	The corrected, converted, calibrated final value of the sensor. see Semantics section
7	Required	Get	V	Status	BYTE	Alarm and Warning State of this object instance	see Semantics section
8	Optional	Set	NV	Alarm Enable	BOOL	Enables the setting of the Alarm Status Bits	0 = disable [default] 1 = enable
9	Optional	Set	NV	Warning Enable	BOOL	Enables the setting of the Warning Status Bits	0 = disable [default] 1 = enable
10	Optional	Get	NV	Full Scale	INT or specified by <i>Data Type</i> if supported	The <i>Value</i> of Full Scale for the sensor.  [default] = maximum allowable value for the <i>Data Type</i>	The value of attribute <i>Value</i> corresponding to the Full Scale calibrated measurement of the sensor.
11	Optional	Get	NV	Offset-A Data Type	USINT	Determines the Data Type of attribute <i>Offset-A</i>	see Semantics section

S-Analog Sensor Object, Class Code: 31<sub>Hex</sub>

Attr ID	Need in Implementation	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
12	Optional	Set	NV	Offset-A	INT or specified by <i>Offset-A Data Type</i> if supported	An amount added prior to <i>Gain</i> to derive <i>Value</i>	see Semantics section 0 = [default]
13	Required if Attribute “Gain” is other than REAL	Get	NV	Gain Data Type	USINT	Determines the Data Type of attribute <i>Gain</i>	see Semantics section
14	Optional	Set	NV	Gain	REAL or specified by <i>Gain Data Type</i> if supported	An amount scaled to derive <i>Value</i>	see Semantics section 1.0 = [default]
15	Required if Attribute “Gain” is other than REAL	Get	NV	Unity Gain Reference	REAL or specified by <i>Gain Data Type</i> if supported	Specifies the value of the <i>Gain</i> attribute equivalent to a gain of 1.0	Used for normalizing the <i>Gain</i> attribute. [default] = 1.0 e.g., for an UINT type Gain, a Unity Gain Reference may be 10000, allowing a gain of 0.0001 to 6.5535.
16	Optional	Set	NV	Offset-B	INT or specified by <i>Data Type</i> if supported	An amount added to derive <i>Value</i>	see Semantics section 0 = [default]
17	Optional	Set	NV	Alarm Trip Point High	INT or specified by <i>Data Type</i> if supported	Determines the Value above which an Alarm Condition will occur	see Semantics section [default] = Maximum value for its data type.
18	Optional	Set	NV	Alarm Trip Point Low	INT or specified by <i>Data Type</i> if supported	Determines the Value below which an Alarm Condition will occur	see Semantics section [default] = Minimum value for its data type.
19	Optional	Set	NV	Alarm Hysteresis	INT or specified by <i>Data Type</i> if supported	Determines the amount by which the <i>Value</i> must recover to clear an Alarm Condition	see Semantics section [default] = 0
20	Optional	Set	NV	Alarm Settling Time	UINT	Determines the time that the <i>Value</i> must exceed the Trip Point before the exception condition is generated.	Time in milliseconds see Semantics section [default] = 0
21	Optional	Set	NV	Warning Trip Point High	INT or specified by <i>Data Type</i> if supported	Determines the <i>Value</i> above which a Warning Condition will occur	see Semantics section [default] = Maximum value for its data type.

S-Analog Sensor Object, Class Code: 31<sub>Hex</sub>

Attr ID	Need in Implementation	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
22	Optional	Set	NV	Warning Trip Point Low	INT or specified by Data Type if supported	Determines the Value below which a Warning Condition will occur	see Semantics section [default] = Minimum value for its data type.
23	Optional	Set	NV	Warning Hysteresis	INT or specified by Data Type if supported	Determines the amount by which the Value must recover to clear a Warning Condition	see Semantics section [default] = 0
24	Optional	Set	NV	Warning Settling Time	UINT	Determines the time that the Value must exceed the Trip Point before the exception condition is generated.	Time in milliseconds see Semantics section [default] = 0
25	Optional	Set	NV	Safe State	USINT	Specifies the behavior for the Value for states other than Execute	see Semantics section [default] = 0
26	Optional	Set	NV	Safe Value	INT or specified by Data Type if supported	The Value to be used for Safe State = Safe Value	see Semantics section [default] = 0
27	Optional	Set	NV	Autozero Enable	BOOL	Enables the Autozero	see Semantics section 0 = disable [default] 1 = enable
28	Optional	Get	V	Autozero Status	BOOL	Indicates the status of the automatic nulling	see Semantics section [default] = 0
29	Optional	Set	NV	Autorange Enable	BOOL	Enables the automatic range switching	see Semantics section 0 = disable [default] 1 = enable
30	Optional	Get	V	Range Multiplier	REAL	Indicates the current range multiplier	see Semantics section [default] = 1.0
31	Optional	Set	NV	Averaging Time	UINT	Specifies the time over which analog samples are averaged.	Time in Milliseconds of a moving-window average. 0 = disable averaging [default] Values less than the sample rate of the device also disable averaging.
32	Optional	Get	NV	Overrange	INT or specified by Data Type if supported	Specifies the highest valid Value	The value above which attribute Reading Valid is set to invalid. [default] = maximum allowable value for the Data Type

S-Analog Sensor Object, Class Code: 31<sub>Hex</sub>

Attr ID	Need in Implementation	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
33	Optional	Get	NV	Underrange	INT or specified by <i>Data Type</i> if supported	Specifies the lowest valid <i>Value</i>	The value below which attribute <i>Reading Valid</i> is set to invalid. [default] = minimum allowable value for the <i>Data Type</i>
34	Optional	Set	NV	Produce Trigger Delta	INT or specified by <i>Data Type</i> if supported <i>and/or Produce Trigger Delta Type</i>	The amount by which <i>Value</i> must change before a Change of State Production is triggered	0 = Disabled [default] See Semantics section
35	Conditional *	Set	NV	Calibration Object Instance	UINT	Indicates which Calibration object instance is active for this object	0 = Disabled [default] See Semantics section
36	Optional	Set	NV	Produce Trigger Delta Type	USINT	Specifies the Type for Produce Trigger Delta	0 = [default] See Semantics section
37	Optional	Set	NV	Value Descriptor	SHORT STRING	A user defined ASCII value used to identify the sensed parameter	Maximum of 20 characters
79-96	Defined by Subclasses below						
97-98	Reserved by CIP for Future Use						
99	Conditional **	Get	NV	Subclass	UINT	Identifies a subset of additional instance attributes, services and behaviors.	0 = No subclass 1 = Flow Diagnostics 2 = Heat Transfer Vacuum Gauge 3 = Capacitance Manometer 4 = Cold Cathode Ion Gauge 5 = Hot Cathode Ion Gauge 6 = Transfer Function 7-65535=Reserved by CIP

\* See Semantics section.

\*\* If the value of Subclass is 00, then this attribute is OPTIONAL in implementation, otherwise, this attribute is REQUIRED.

### 5-36.2.1 Subclasses

Each instance level subclass defines a unique meaning for an overlapping range of instance attribute IDs and/or instance service IDs. The range for subclass definitions begins at ID 96 and numbers downward for attributes, and ID 63<sub>hex</sub> and numbers downward for services. The subclass for a given instance is identified by the value of its Subclass instance attribute. The subclasses for this object are specified at the end of this object specification section.

### 5-36.3 Semantics

#### 5-36.3.1 Data Type

All Data Type attributes, including *Data Type*, *Offset-A Data Type* and *Gain Data Type*, use the enumerated values specified in Appendix C-6.1.

The *Data Type* attribute is settable only in the *Idle State* and only if no attribute belonging to the object instance is the endpoint of an I/O connection in the *Established State*.

The *Data Type* attribute may change automatically based upon established I/O connections. See Behavior section for more information on this mechanism.

#### 5-36.3.2 Data Units

Specifies the context of *Value* and related attributes (such as, offset and trip points) for this object instance. See Appendix D for a list of values. A request to set attribute to an unsupported value will return an error response.

The *Data Units* attribute is settable only in the *Idle State*.

#### 5-36.3.3 Value, Offset (A and B) and Gain

An S-Analog Sensor object instance derives a reading from a physical analog sensor. The reading is converted to the data type and units specified for the *Value* attribute. The *Offset-A*, *Offset-B* and *Gain* attributes are applied to the sensor reading as specified by the following formula:

$$\text{Value} = \text{Gain} \bullet (\text{Sensor Reading} + \text{Offset-A}) + \text{Offset-B}$$

Typically, the *Offset-A* or *Offset-B* attributes are modified by the Zero-Adjust service and the *Gain* attribute is modified by the Gain\_Adjust services; particularly, when the device utilizes a non-linear conversion algorithm. However, support of these services is not required. See Behavior section.

#### 5-36.3.4 Status

A bit mapped byte which indicates the Alarm and Warning Exception status of the object instance. The following definition applies:

**Table 5-36.3 Status Attribute Bit Definitions**

Bit	Definition
0	High Alarm Exception: 0 = cleared; 1 = set
1	Low Alarm Exception: 0 = cleared; 1 = set
2	High Warning Exception: 0 = cleared; 1 = set
3	Low Warning Exception: 0 = cleared; 1 = set
4	Reserved
5	Reserved
6	Reserved
7	Reserved

### 5-36.3.5 Trip Points, Hysteresis and Settling Time

Trip Point High is the level above which the *Value* attribute will cause an Alarm or Warning exception condition.

Trip Point Low is the level below which the *Value* attribute will cause an Alarm or Warning exception condition.

A Hysteresis value specifies the amount by which the Value attribute must transition in order to clear an Alarm or Warning condition. For example: A Trip Point High value of 100 and a hysteresis value of 2 will result in an exception condition being set when the Value is above 100 and cleared when the Value drops below 98. Similarly, A Trip Point Low value of 100 and a hysteresis value of 2 will result in an exception condition being set when the Value is below 100 and cleared when the Value increases above 102.

The Settling Time determines the amount of time that the Value attribute must exceed the Trip Point before the exception condition is generated. The Settling Time also applies to the clearing of the condition.

### 5-36.3.6 Safe State

This attribute specifies what value will be held in *Value* for states other than Executing. See the S-Device Supervisor object definition in Section 5-35. for a description of object states. The purpose of this mechanism is to allow other devices, that may be using this *Value*, to transition to, or remain in, a safe state in the event of this device transitioning to a FAULT, IDLE, or ABORT state. The following values are defined:

**Table 5-36.4 Safe State Attribute Values**

Attribute Value	State
0	Zero
1	Full Scale
2	Hold Last Value
3	Use Safe Value
4	Continue Sensing
5-50	Reserved
51-99	Device Specific
100-255	Vendor Specific

### 5-36.3.7 Safe Value

For Safe State set to Use Safe Value, this attribute holds the value to which the *Value* attribute will be set for object instance states other than Executing.

### 5-36.3.8 Autozero Enable and Autozero Status

When the autozero is enabled, the device will automatically invoke a Zero\_Adjust service request (no parameter) contingent upon a set of conditions specified by the manufacturer. These conditions may be determined by the value of an attribute (e.g., setpoint) or some other mechanism defined by the manufacturer. See Zero\_Adjust service.

While the device is in the process of nulling due to an Autozero event, the value of *Autozero Status* is one (1). When the device is not in the process of nulling, this value is zero (0).

### 5-36.3.9 Autorange Enable and Range Multiplier

When the autorange is enabled, the device will automatically switch full-scale range based on a set of conditions specified by the manufacturer. The Range Multiplier indicates the range scale.

An example of how Autorange may work is: when the *Value* is less than 9% with a *Range Multiplier* of 1.0, the *Range Multiplier* switches to 10.0 (the *Value* then reads 90% of the 10X range). When the *Value* then reaches 100% with a *Range Multiplier* of 10.0, the *Range Multiplier* returns to 1.0 (the *Value* then reads 10% of the 1X range).

### 5-36.3.10 Produce Trigger Delta

This attribute is used in conjunction with the "Change of State" production trigger type. Upon transition of the associated connection object instance (any Change of State connection pointing to the S-Analog Sensor object *Value* attribute) to the established state, a production is immediately triggered and this reported *Value* is stored internally for the determination of the next production trigger. When the *Value* changes by an amount of at least the *Produce Trigger Delta* (i.e., the *Value* as compared to the internally stored previously produced *Value*), a new production is triggered, and this reported *Value* becomes the new internally stored *Value* for the determination of the next production trigger. The interpretation of this attribute is absolute value unless otherwise specified by the *Produce Trigger Delta Type* attribute.

### 5-36.3.11 Calibration Object Instance

This attribute is used to select an instance of the S-Gas Calibration object or the S-Sensor Calibration object as specified by the device profile where this object is used. The selected S-Calibration object instance provides the data with which an S-Analog Sensor object instance enacts the appropriate calibration algorithm for a given gas type, material type or other parameter dependency that might affect a sensor's calibration.

A Set\_Attribute\_Single request, specifying a value not supported, will return an "invalid attribute value" error response. A list of acceptable values for this attribute is derived from a class level service request to the Calibration object.

\* Conditionally Required: If a device profile specifies a Calibration object relationship for an S-Analog Sensor object instance, then this attribute is required. See the S-Gas Calibration object or the S-Sensor Calibration object definition for more information.

### 5-36.3.12 Produce Trigger Delta Type

The *Produce Trigger Delta Type* attribute specifies the interpretation of the *Produce Trigger Delta* attribute. The following values are defined:

**Table 5-36.5 Produced Trigger Delta Type Attribute Values**

Produce Trigger Delta Type Attribute Value	Produce Trigger Delta Type	Produce Trigger Delta Data Type
0 [default]	Absolute Value	As Specified by Data Type
1	Percent (see below)	UINT
2 - 255	Reserved by CIP for Future Use	

Due to the logarithmic, or other non-linear nature of some analog measurements, the data units of the *Produce Trigger Delta* attribute may be set to a Percent *Produce Trigger Delta*. This also causes the Data Type of *Produce Trigger Delta* to be type UINT with a resolution of 0.1%.

For example, if the last COS-produced reading was 5E-5, the *Produce Delta Trigger Type* was set to 1 (Percent), and the *Produce Trigger Delta* = 200 (which equates to a delta of 20%), then the next reading will be produced at 4E-5 or 6E-5.

### 5-36.3.13 Value Descriptor

The Value Descriptor attribute identifies the Value of the S-Analog Sensor Object. It describes the physical meaning of the value the sensor delivers, e.g.: ion current, electron density or intensity of one wavelength of an optical emission spectrum.

## 5-36.4 Common Services

The S-Analog Sensor Object provides the following Common Services:

**Table 5-36.6 S-Analog Sensor Object Common Services**

Service Code	Need in Implementation		Service Name	Description of Service
	Class	Instance		
0E <sub>hex</sub>	Conditional *	Required	Get_Attribute_Single	Returns the contents of the specified attribute.
01 <sub>hex</sub>	n/a	Required	Set_Attribute_Single	Modifies an attribute value.

\*The Get\_Attribute\_Single service is REQUIRED if any attributes are implemented.

See Appendix A for definition of these services

## 5-36.5 Object-specific Services

The S-Analog Sensor Object provides the following Object-Specific Services:

**Table 5-36.7 S-Analog Sensor Object Object-specific Services**

Service Code	Need in Implementation		Service Name	Description of Service
	Class	Instance		
4B <sub>hex</sub>	n/a	Optional	Zero_Adjust	Causes the device to modify attribute <i>Offset-A</i> and/or <i>Offset-B</i> such that attribute <i>Value</i> equals the Target Value sent with the request.
4C <sub>hex</sub>	n/a	Optional	Gain_Adjust	Causes the device to modify attribute <i>Gain</i> , such that attribute <i>Value</i> , equals the Target Value sent with the request.

### 5-36.5.1 Zero\_Adjust and Gain\_Adjust Services

The Zero\_Adjust and Gain\_Adjust services are used to cause the S-Analog Sensor Object device to modify its *Offset-A* and/or *Offset-B* and *Gain* attribute values based upon manufacturer specific algorithms. The target value specified in the service request represents the actual parametric measurement that the physical sensor should be reporting at the time of the request.

There are no state transitions associated with the invocation of these services. It is, therefore, incumbent upon the user to establish the device into the desired configuration prior to, and during, the execution of these services. This will generally involve exposing the sensor to a known environment and treating the values read during execution of the services accordingly.

A success service response indicates that the service was accepted and the application process started.

**Table 5-36.8 Zero\_Adjust Request Service Data Field Parameters**

Parameter	Required	Data Type	Description	Semantics of Values
Target Value	Optional	Specified by the value of attribute <i>Data Type</i>	The target value for the zero calibration	The value to which the <i>Value</i> attribute will be set. If not specified, the default value of zero is used.

**Table 5-36.9 Gain\_Adjust Request Service Data Field Parameters**

Parameter	Required	Data Type	Description	Semantics of Values
Target Value	Required	Specified by the value of attribute <i>Data Type</i>	The target value for the gain calibration	The value to which the <i>Value</i> attribute will be set.

## 5-36.6 Behavior

The behavior of this object is managed by the S-Device Supervisor Object in Section 5-35.5.

An S-Analog Sensor object instance acquires a reading from a physical sensor, as identified by the application of the object, and applies an algorithm to modify the reading into the appropriate *Date Type* and *Data Units*. Optionally, additional corrective algorithms are applied to further correct for various calibration effects. These additional algorithms are specified in other objects, as identified in the device profile, or as extensions, specified by the manufacturer.

All Full Scale, Trip Point, Overrange and Underrange calculations, as specified above, utilize the *Value* attribute.

### **5.36.6.1 Data Type**

If the implementation of this object specifies more than one valid Data Type value, in the device profile or by vendor, then the following behavior with respect to *Data Type* applies: The Data Type value will be set automatically based upon the first valid I/O connection established by the device. If, however, a device is specified by a vendor to support only one Data Type, this behavior is not supported.

If no established I/O connections exist, which include an attribute from this object, then the *Data Type* attribute is settable provided that the object is in the *Idle State*. The following example demonstrates this behavior:

A device profile specifies an instance of the S-Analog Sensor object as well as two static Assembly object instances, both with data attribute components mapped to this object instance. Assembly object instance ID 1 specifies INT data types and Assembly object instance ID 2 specifies REAL data types.

After the device is On-Line, it is configured with an I/O connection to Assembly instance ID 2. When the connection transitions to the *Established State*, this object instance attribute *Data Type* is automatically set with the value for REAL before any data is communicated to, or from, the object instance. Any subsequent attempt to connect to Assembly instance ID 1 would then be rejected and result in an INVALID ATTRIBUTE VALUE error with the additional error code indicating the ID of the offending attribute, which in this case would be the connection path.

## 5-36.7 S-Analog Sensor Object Class Subclass 01 (Instance Selector)

The following is a **class-level subclass** extension to the S-Analog Sensor Object. It provides a single port mechanism for flowing data from the active S-Analog Sensor Instance and allowing that the Active Instance ID may change. The intended application is for Gauge Controllers where more than one gauge is connected, but only one gauge is considered the "active" gauge.

### 5-36.7.1 Class Attributes

The following Class Attributes are specified for this object subclass.

**Table 5-36.10 Instance Selector Subclass Class Attributes**

Attr ID	Need in Implementation	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
94	Optional	Get	V	Active Value	INT or specified by Data Type (Instance Attribute ID 3) if supported	Can be used by assemblies to produce this class-level attribute, instead of the Value (Attribute ID 6) of the S-Analog Sensor Instances.	
95	Optional	Get	V	Active Instance Number	UINT	Identifies the object instance that is providing the <i>Value</i> which is copied into the <i>Active Value</i> for all input Assemblies and the Alarm/Warning Exception Details for the S-Device Supervisor object.  See Behavior section.	Default = 1
96	Optional	Get	NV	Number of Gauges	USINT	Identifies the number of gauge instances present in the device.	default = [gauge-specific]

### 5-36.7.2 Semantics

#### 5-36.7.2.1 Active Value

Assemblies or connections may produce this class-level attribute, instead of the *Value* (Attribute ID 6) of the active S-Analog Sensor instance. The S-Analog Sensor class-level attribute *Active Instance Number* identifies the object instance that is currently active and providing the *Value* to the *Active Value* class-level attribute which is, in turn, produced by the input assemblies that have *Active Value* as a member.

#### 5-36.7.2.2 Active Instance Number

The device internally modifies this attribute, as required, to identify the S-Analog Sensor object instance providing the *Value* member which is copied into the *Active Value* for all Input Assemblies and the Alarm/Warning Exception Details for the S-Device Supervisor object.

See Behavior for more information on the mechanism.

#### **5-36.7.2.3 Number of Gauges**

This attribute is used to determine the size of all Input Assemblies within a node. See the respective Device Profile for its usage within a Device Type.

#### **5-36.7.3 Services**

There are no additions or restrictions to the Object Services for this object subclass.

#### **5-36.7.4 Behavior**

The specific algorithm used to set the *Active Instance Number* is manufacturer specific. However, most will follow the basic logic that each of multiple gauges are valid (or ideally suited) for specific ranges of measurement. Therefore, the *Active Instance Number* will be modified based upon the *Active Value* in order that the best gauge, corresponding to a given S-Analog Sensor instance, will be active for the given measurement range.

## 5-36.8 S-Analog Sensor Object Instance Subclass 01 (Flow Diagnostics)

The following specification applies to a subclass of this object for application in Mass Flow Controller devices.

### 5-36.8.1 Instance Attributes

The following Instance Attributes are specified for this object subclass.

**Table 5-36.11 Flow Diagnostics Subclass Instance Attributes**

Attr ID	Need in Implementation	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
95	Optional	Set	NV	Flow Totalizer	ULINT	Total gas flowed through the device since this value was last set to zero	Units are Standard CCs see Behavior section default = 0
96	Optional	Set	NV	Flow Hours	UDINT	Total time device has been powered and flowing gas since this value was last set to zero	Resolution is one hour see Behavior section default = 0

### 5-36.8.2 Services

There are no additions or restrictions to the Object Services for this object subclass.

### 5-36.8.3 Behavior

#### 5-36.8.3.1 Flow Totalizer and Flow Hours Process

The factory configured out-of-box values for the Flow Totalizer and Flow Hours attributes are both zero. The attributes are only modifiable with *set\_attribute\_single* service requests; they are not altered by the *Reset* service, including power-cycle, of either the Identity or the S-Device Supervisor objects.

The Flow Totalizer attribute is incremented, at a rate of once every cubic centimeter of gas flow, by the S-Analog Sensor object instance to reflect the amount of gas that has flowed through the device. Upon reaching its maximum value, the Flow Totalizer value is no longer incremented and remains at its maximum value.

The Flow Hours attribute is incremented, at a rate of once every hour, by the S-Analog Sensor object instance to reflect the amount of time that gas has flowed through the device. This condition is determined by the *Value* attribute being greater than 0.5% of full scale. Upon reaching its maximum value, the Flow Hours value is no longer incremented and remains at its maximum value.

## 5-36.9 S-Analog Sensor Object Instance Subclass 02 (Heat Transfer Vacuum Gauge)

Following is a subclass extension to the S-Analog Sensor Object. The original usage for the subclass is for Heat Transfer Vacuum Gauges. The Heat Transfer Vacuum Gauge extension is used in conjunction with the Vacuum/Pressure Gauge Profile providing control and status information unique to heat transfer gauges - Convection, Pirani and Thermocouple gauge types.

### 5-36.9.1 Instance Attributes

Certain minimal implementations may support any optional "Set" attributes as "Get" only and still be compliant with this object specification. Although no attribute is individually required, this object requires that one or more attributes be used.

**Table 5-36.12 Heat Transfer Vacuum Gauge Subclass Instance Attributes**

Attr ID	Need in Implementation	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
91	Optional	Set	NV	Cable Length	UINT	Transducer cable length for compensation, if required	[default = 0] Units in cm see Semantics
92	Optional	Set	NV	Sensor High Temp Warning Value	REAL	Maximum compensatable Sensor Temperature.	[default = 0] Units in °C
93	Optional	Get	V	Sensor Temperature	REAL	Sensor temperature in °C.	see Semantics
94	Optional	Get	V	Sensor Warning	STRUCT of BYTE BYTE	Bit definitions of Sensor Warning	[default=0] see Semantics
95	Optional	Get	V	Sensor Alarm	STRUCT of BYTE BYTE	Bit definitions of Sensor Alarms	[default=0] see Semantics
96	Optional	Get	V	Status Extension	BYTE	Bit-mapped byte providing additional status bits of an S-Analog Sensor instance.	[default=0] see Semantics

### 5-36.9.2 Semantics

#### 5-36.9.2.1 Cable Length

This attribute specifies the cable length of the transducer cable (cable between the sensor [filament] and its electronic).

#### 5-36.9.2.2 Sensor Temperature

Ambient temperature of the sensor housing. This value could be used for temperature compensation.

### 5-36.9.2.3 Sensor Warning

The Sensor Warning attribute provides 16 bits for the current Sensor Warning state of the object. The Sensor Warning attribute also maps to the S-Device Supervisor Exception Detail Warning attribute. The following table provides definitions for each Sensor Warning bit. Reserved bits must be set to zero.

**Table 5-36.13 Sensor Warning Attribute**

Data Component	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Sensor Warning Byte 0	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
Sensor Warning Byte 1	High Warning Exception	Low Warning Exception	Reserved	Reserved	Reserved	Sensor High Temperature Warning	Electronics Warning	Reserved

### 5-36.9.2.4 Sensor Alarm

The Sensor Alarm attribute provides 16 bits for the current Sensor Alarm state of the object. The Sensor Alarm attribute also maps to the S-Device Supervisor Exception Detail Alarm attribute. The following table provides definitions for each Sensor Alarm bit. Reserved bits must be set to zero.

**Table 5-36.14 Sensor Alarm Attribute**

Data Component	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Sensor Alarm Byte 0	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Sensor Element Failure
Sensor Alarm Byte 1	High Alarm Exception	Low Alarm Exception	Reserved	Reserved	Reserved	Over Temperature of Electronics	Electronics Failure	Reserved

### 5-36.9.2.5 Status Extension

8 Bits providing the current sensor alarm state of the object.

**Table 5-36.15 Status Extension Attribute**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	Reserved	Reserved	Reserved	Reserved	Underrange Exceeded	OVERRANGE EXCEEDED	READING INVALID*

\* Note: Logical inversion of Reading Valid

### 5-36.9.3 Services

There are no additions or restrictions to the Object Services for this object subclass.

### 5-36.9.4 Behavior

There are no additions or restrictions to the Object Behavior for this object subclass.

## 5-36.10 S-Analog Sensor Object Instance Subclass 03 (Diaphragm Gauge)

Following is a subclass extension to the S-Analog Sensor Object. The original usage for the subclass is for Diaphragm Gauge Gauges. The Diaphragm Gauge extension is used in conjunction with the Vacuum/Pressure Gauge Profile providing control and status information.

### 5-36.10.1 Instance Attributes

Certain minimal implementations may support any optional "Set" attributes as "Get" only and still be compliant with this object specification. Although no attribute is individually required, this object requires that one or more attributes be used.

**Table 5-36.16 Diaphragm Gauge Subclass Instance Attributes**

Attr ID	Need in Implementation	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
93	Optional	Set	NV	Sensor Temperature	REAL	The temperature in Celsius at which the sensor has warmed up.	degrees Celsius [default] = vendor specific
94	Optional	Get	V	Sensor Warning	STRUCT of BYTE BYTE	Bit definitions of Sensor Warnings	[default=0] See Semantics
95	Optional	Get	V	Sensor Alarm	STRUCT of BYTE BYTE	Bit definitions of Sensor Alarms	[default=0] See Semantics
96	Optional	Get	V	Status Extension	BYTE	Bit mapped byte providing additional status of an S-Analog Sensor instance.	[default=0] See Semantics

### 5-36.10.2 Semantics

#### 5-36.10.2.1 Sensor Warning

The Sensor Warning attribute provides 16 bits for the current Sensor Warning state of the object. The Sensor Warning attribute also maps to the S-Device Supervisor Exception Detail Warning attribute as specified in the Vacuum/Pressure Gauge device profile. The following table provides definitions for each Sensor Warning bit. Reserved bits must be set to zero.

**Table 5-36.17 Sensor Warning Attribute**

Data Component	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Sensor Warning Byte 0	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Not At Temperature
Sensor Warning Byte 1	High Warning Exception	Low Warning Exception	Reserved	Reserved	Reserved	Reserved	Electronics Warning	Reserved

### 5-36.10.2.2 Sensor Alarm

The Sensor Alarm attribute provides 16 bits for the current Sensor Alarm state of the object. The Sensor Alarm attribute also maps to the S-Device Supervisor Exception Detail Alarm attribute as specified in the Vacuum/Pressure Gauge device profile. The following table provides definitions for each Sensor Alarm bit. Reserved bits must be set to zero.

**Table 5-36.18 Sensor Alarm Attribute**

Data Component	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Sensor Alarm Byte 0	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Diaphragm Failure
Sensor Alarm Byte 1	High Alarm Exception	Low Alarm Exception	Reserved	Reserved	Reserved	Over Temperature of Electronics	Electronics Failure	Reserved

### 5-36.10.2.3 Status Extension

8 Bits providing the current sensor alarm state of the object.

**Table 5-36.19 Status Extension Attribute**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	Reserved	Reserved	Reserved	Reserved	Underrange Exceeded	Overrange Exceeded	Reading Invalid*

\* Note: Logical inversion of Reading Valid

### 5-36.10.3 Services

There are no additions or restrictions to the Object Services for this object subclass.

### 5-36.10.4 Behavior

For heated sensors, the *Sensor Temperature* attribute indicates the temperature at which the sensor has warmed up. Once the sensor has "warmed up", the *Reading Valid* attribute of the S-Analog Sensor Object will be allowed to be set to TRUE provided that all other conditions governing its behavior are also met. The warning bit in the Sensor Warning attribute for *Not At Temperature* shall be cleared.

## 5-36.11 S-Analog Sensor Object Instance Subclass 04 (Cold Cathode Ion Gauge)

Following is a subclass extension to the S-Analog Sensor Object. The original usage for the subclass is for Cold Cathode Ion Vacuum Gauges. The Cold Cathode Ion Vacuum Gauge extension is used in conjunction with the Vacuum/Pressure Gauge Profile providing control and status information.

A Cold Cathode Ion Vacuum Gauge is a device that measures pressure at vacuum levels. At low pressures the operating of these gauges depends upon the establishment of a gas discharge between two metal electrodes by the application of a sufficiently high d.c. voltage. The gas discharge current is pressure dependent.

### 5-36.11.1 Instance Attributes

Certain minimal implementations may support any optional "Set" attributes as "Get" only and still be compliant with this object specification.

**Table 5-36.20 Cold Cathode Ion Guage Subclass Instance Attributes**

Attr ID	Need in Implementation	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
91	Optional	Set	V	High Voltage	REAL	High voltage applied to the anode.	Volts [default] = vendor specific
92	Optional	Set	NV	Sensitivity	REAL	Conversion Factor from current ratio to pressure	[default] = 1.0 see Semantics section
93	Required	Get	V	High Voltage Status	BOOL	Indicates whether the high voltage is turned on or off.	0 = Off [default] 1 = On
94	Optional	Get	V	Sensor Warning	STRUCT of BYTE BYTE	Bit definitions of Sensor Warnings	[default=0] see Semantics
95	Optional	Get	V	Sensor Alarm	STRUCT of BYTE BYTE	Bit definitions of Sensor Alarms	[default=0] see Semantics
96	Optional	Get	V	Status Extension	BYTE	Bit mapped byte providing additional status of an S-Analog Sensor instance	[default=0] see Semantics

### 5-36.11.2 Semantics:

#### 5-36.11.2.1 High Voltage

Valid values for the High Voltage attribute are manufacturer specific. Attempts to set the High Voltage to an invalid value shall return the Invalid Attribute Value error response.

#### 5-36.11.2.2 Sensitivity

The conversion factor used to convert the ratio of gauge currents to pressure. This conversion is manufacturer specific.

### 5-36.11.2.3 Sensor Warning

The Sensor Warning attribute provides 16 bits for the current Sensor Warning state of the object. The Sensor Warning attribute also maps to the S-Device Supervisor Exception Detail Warning attribute as specified in the Vacuum/Pressure Gauge device profile. The following table provides definitions for each Sensor Warning bit. Reserved bits must be set to zero.

**Table 5-36.21 Sensor Warning Attribute**

Data Component	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Sensor Warning Byte 0	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
Sensor Warning Byte 1	High Warning Exception	Low Warning Exception	Reserved	Reserved	Overpressure High Voltage Off	No Ignition Detected	Electronics Warning	Reserved

### 5-36.11.2.4 Sensor Alarm

The Sensor Alarm attribute provides 16 bits for the current Sensor Alarm state of the object. The Sensor Alarm attribute also maps to the S-Device Supervisor Exception Detail Alarm attribute as specified in the Vacuum/Pressure Gauge device profile. The following table provides definitions for each Sensor Alarm bit. Reserved bits must be set to zero.

**Table 5-36.22 Sensor Alarm Attribute**

Data Component	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Sensor Alarm Byte 0	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
Sensor Alarm Byte 1	High Alarm Exception	Low Alarm Exception	Reserved	Reserved	Overpressure High Voltage Off	Over Temperature of Electronics	Electronics Failure	Reserved

### 5-36.11.2.5 Status Extension

8 Bits providing the current sensor alarm state of the object.

**Table 5-36.23 Status Extension Attribute**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	Reserved	Reserved	Reserved	Reserved	Underrange Exceeded	Overrange Exceeded	Reading Invalid*

\* Note: Logical inversion of Reading Valid

### 5-36.11.3 Services

The following instance-level services are defined for this subclass of the S-Analog Sensor object:

**Table 5-36.24 Cold Cathode Ion Gauge Subclass Instance Services**

Service Code	Need in Implementation		Service Name	Description of Service
	Class	Instance		
62 <sub>hex</sub>	n/a	Conditional *	Set High Voltage State	Sets the high voltage to the state specified by the High Voltage State parameter.
63 <sub>hex</sub>	n/a	Conditional *	Clear Overpressure High Voltage Off Alarm	Clears the Overpressure High Voltage Off bit after an automatic high voltage off

\* Required for single gauges only, otherwise support is optional

### 5-36.11.3.1 Clear Overpressure High Voltage Off Alarm Service

If the high voltage is switched off automatically in case of an overpressure condition, it is not possible to switch on the high voltage again before the Overpressure High Voltage Off bit is reset. This service attempts to reset the Overpressure High Voltage Off bit. See the State Event Matrix below.

### 5-36.11.3.2 Set High Voltage State Service

Attempts to set the high voltage to the state specified by the HighVoltageState parameter. This service may fail depending upon the current object state. See the State Event Matrix below.

**Table 5-36.25 Set High Voltage State Service Data Field Parameters**

Parameter	Required	Data Type	Description	Semantics of Values
HighVoltageState	Required	BOOL	See Behavior	0 = switches high voltage OFF 1 = switches high voltage ON

### 5-36.11.4 Behavior

#### 5-36.11.4.1 Alarms/Warnings

If any bit of the attribute "Sensor Alarm" is set, the high voltage is switched off automatically.

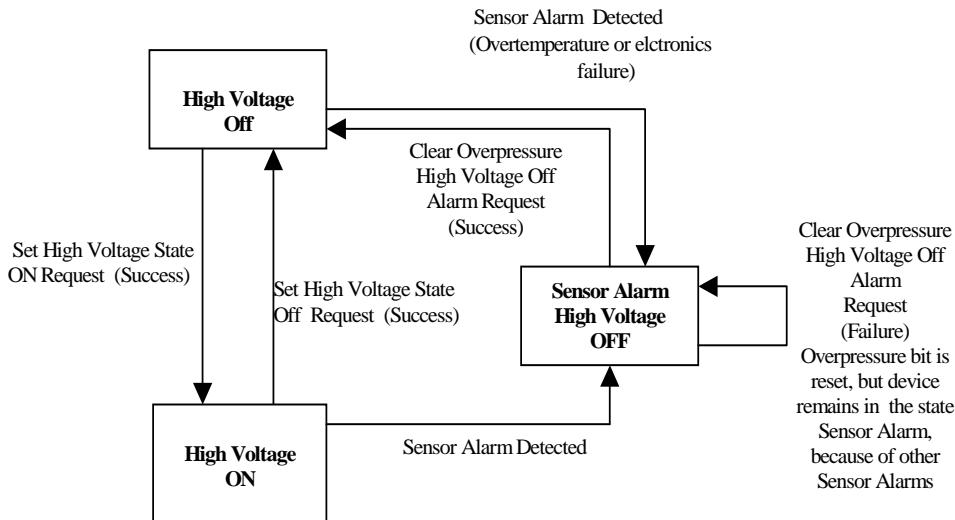
#### 5-36.11.4.2 No ignition detection

The "No Ignition Detected" bit in the Sensor Warning attribute will be set under the following conditions:

- The discharge is undetectable during normal operation.
- Ignition is not detected following an Explicit request to set the High Voltage ON attribute to On.

These conditions are likely to arise when the gauge is operating at very low pressures.

Figure 5-36.26 S-Analog Sensor Object Instance Behavior



See the S-Device Supervisor object (Chapter 5-35) for more information on the Executing State.

Table 5-36.27 S-Analog Sensor Sub-State Event Matrix

EVENT	SUB – STATE		
	High Voltage OFF	High Voltage ON	Sensor Alarm High Voltage Off
Set High Voltage State ON Request	Transition to <i>High Voltage ON</i>	Error AIRS *	Error OSC **
Set High Voltage State OFF Request	Error AIRS *	Transition to <i>High Voltage OFF</i>	Error OSC **
Electronics Failure	Transition to <i>Sensor Alarm High Voltage Off</i>	Transition to <i>Sensor Alarm High Voltage Off</i>	Event reported in Sensor Alarm
Overtemperature	Transition to <i>Sensor Alarm High Voltage Off</i>	Transition to <i>Sensor Alarm High Voltage Off</i>	Event reported in Sensor Alarm
Clear Overpressure High Voltage Off Alarm Request (Condition: ONLY Overpressure High Voltage Off in Sensor Alarm set)	Not Applicable	Not Applicable	Transition to <i>High Voltage OFF</i>
Clear Overpressure High Voltage Off Alarm Request (Condition: Overpressure Emission Off and other Sensor Alarms set)	Not Applicable	Not Applicable	Error OSC **
Clear Overpressure High Voltage Off Alarm Request (Sensor Alarms not set)	Error AIRS *	Error OSC **	Not Applicable

\* Error AIRS = Error Response “Already in Requested Mode/State” (Code 0B<sub>hex</sub>)

\*\* Error OSC = Error Response “Object State Conflict” (Code 0C<sub>hex</sub>)

## 5-36.12 S-Analog Sensor Object Instance Subclass 05 (Hot Cathode Ion Gauge)

Following is a subclass extension to the S-Analog Sensor Object. The original usage for the subclass is for Hot Cathode Ion Vacuum Gauges. The Hot Cathode Ion Vacuum Gauge extension is used in conjunction with the Vacuum/Pressure Gauge Profile providing control and status information.

A Hot Cathode Ion Vacuum Gauge is a device that measures pressure at vacuum levels. It contains a hot filament for the generation of free electrons which are accelerated (emission current) and, by collision, generate ionized gas molecules. The positive ionized molecules are attracted to, and collected by, a negatively charged collector (collector current). The ratio of the two currents (collector to emission) is proportional to the pressure.

Generally, these devices support a "degas" mode of operation by a resistive method or an electron bombardment method.

### 5-36.12.1 Instance Attributes

Certain minimal implementations may support any optional "Set" attributes as "Get" only and still be compliant with this subclass specification. All required attributes must be supported as specified.

**Table 5-36.28 Hot Cathode Ion Gauge Subclass Instance Attributes**

Attr ID	Need in Implementation	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
79	Optional	Get	V	Filament Bias Voltage	REAL	Diagnostic readout of filament bias voltage	Volts [default] = vendor specific
80	Optional	Set	V	Grid Voltage	REAL	Setting of grid voltage.	Volts [default] = vendor specific
81	Conditional See Semantics	Set	V	Active Degas Filament	BYTE	Indicates which degas filament is selected.	Bit mapped byte of up to 8 filaments as specified by the Vendor. See Semantics
82	Optional	Set	NV	Degas Power	REAL	Indicates degas power in watts	Power in Watts [default] = vendor specific
83	Optional	Get	V	Filament Current	REAL	Indicates actual filament current in A	Filament Current in amps. [default] = vendor specific
84	Optional	Get	V	Degas Time Off Remaining	UINT	Remaining time, in seconds, for the current degas off cycle.	[default] = 0 See Semantics
85	Optional	Set	NV	Degas Time Off Interval	UINT	Minimum time, in seconds, between consecutive degas functions.	[default] = vendor specific See Semantics

S-Analog Sensor Object, Class Code: 31<sub>Hex</sub>

Attr ID	Need in Implementation	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
86	Optional	Get	V	Degas Time On Remaining	UINT	Remaining time, in seconds, for the current degas on cycle.	[default] = 0 See Semantics
87	Optional	Set	V	Degas Time On Interval	UINT	Indicates the length of time, in seconds, that degas is on.	[default] = vendor specific See Semantics
88	Optional	Get	V	Degas Status	BOOL	Indicates current degas state.	0 = Off 1 = On
89	Conditional See Semantics	Set	NV	Active Filament	BYTE	Indicates which filament(s) is/are selected.	Bit mapped byte of up to 8 filaments as specified by the Vendor. See Semantics
90	Optional	Set	NV	Sensitivity	REAL	Conversion Factor from current ratio to pressure	[default] = 1.0 see Semantics section
91	Optional	Set	NV	Emission Current	REAL	Indicates setting level of emission current in amps. Can be indicated in either continuous or discrete readings allowed by the vendor.	Vendor specifies the range of supported values and the [default] Note: Degas Mode may override this setting.
92	Optional	Set	NV	Mode Filament Selection	BOOL	Selects automatic or user filament selection	see Semantics 0 = automatic 1 = user [default]
93	Required	Get	V	Emission Status	BOOL	Indicates whether the emission is turned on or off.	0 = Off 1 = On
94	Optional	Get	V	Sensor Warning	STRUCT of BYTE BYTE	Bit definitions of Sensor Warnings	[default=0] see Semantics
95	Optional	Get	V	Sensor Alarm	STRUCT of BYTE BYTE	Bit definitions of Sensor Alarms	[default=0] see Semantics
96	Optional	Get	V	Status Extension	BYTE	Bit-mapped byte providing additional status bits of an S-Analog Sensor instance.	[default=0] see Semantics

### 5-36.12.2 Semantics

#### 5-36.12.2.1 Active Degas Filament

Indicates which degas filament is active. 0 = off / 1 = on. Filament selection may be made either automatically by the device, or remotely by setting this attribute.

**Table 5-36.29 Active Degas Filament Attribute**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Degas Filament 8	Degas Filament 7	Degas Filament 6	Degas Filament 5	Degas Filament 4	Degas Filament 3	Degas Filament 2	Degas Filament 1

#### 5-36.12.2.2 Degas Time Off Interval and Degas Time Off Remaining

The Degas Time Off Interval attribute specifies the minimum time, in seconds, between degas cycles. The valid range for this attribute is manufacturer specific.

The Degas Time Off Remaining attribute provides the remaining time, in seconds, before a new degas cycle can be started. When the Degas Status attribute transitions to the Off state, the Degas Time Off Remaining attribute is set to the value of the Degas Time Off Interval attribute. The device then decrements the value of the Degas Time Off Remaining attribute at one second intervals until the attribute value reaches zero. The “Object State Conflict” error response will be returned for all Set Degas State ON requests received while the value of the Degas Time Off Remaining attribute is greater than zero.

#### 5-36.12.2.3 Degas Time On Interval and Degas Time On Remaining

The *Degas Time On Interval* attribute specifies the maximum time, in seconds, for degas. The manufacturer may limit the maximum value for this attribute.

The *Degas Time On Remaining* attribute provides the remaining time, in seconds, for the current degas cycle. When the *Degas Status* attribute transitions to the On state, the *Degas Time On Remaining* attribute is set to the value of the *Degas Time On Interval* attribute. The device decrements the value of the *Degas Time On Remaining* attribute at one second intervals and stops the active degas cycle when the attribute value reaches zero. Whenever the *Degas Status* attribute transitions to the Off state from the On state, e.g. alarm detected, Set Degas State request received, etc., the *Degas Time On Remaining* attribute is set to the value zero.

#### 5-36.12.2.4 Active Filament

Indicates which degas filament is active. 0 = off / 1 = on. Filament selection may be made either automatically by the device, or remotely by setting this attribute.

**Table 5-36.30 Active Filament Attribute**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Sensor Filament 8	Sensor Filament 7	Sensor Filament 6	Sensor Filament 5	Sensor Filament 4	Sensor Filament 3	Sensor Filament 2	Sensor Filament 1

### 5-36.12.2.5 Sensitivity

The conversion factor used to convert the ratio of gauge currents to pressure using the following formula:

$$P = (1/S) (C/E)$$

Where:

P = pressure (*Value* attribute)

S = sensitivity (*Sensitivity* attribute)

C = collector current (measured by the device, Amps)

E = emission current (*Emission Current* attribute, Amps)

Sensitivity units are in inverse pressure units.

### 5-36.12.2.6 Mode Filament Selection

The Mode Filament Selection attribute determines whether devices with multiple filaments automatically select the active filament(s) for Emission and Degas ON, or whether the user manually selects the filament(s).

#### 5-36.12.2.6.1 Automatic Filament Selection

With “Automatic Filament Selection” (*Mode Filament Selection* attribute = 0), the device determines, using a manufacturer specific algorithm, which filament to use for Emission ON and Degas ON. When the device is configured for automatic filament selection, the *Active Filament* and *Active Degas Filament* attributes have Get access.

If a filament(s) is broken, the corresponding bit(s) in the *Sensor Warning* attribute is set, and the device selection algorithm must avoid selecting the broken filament(s).

#### 5-36.12.2.6.2 User Filament Selection

With “User Filament Selection” (*Mode Filament Selection* attribute = 1), the user selects which filament to use for Emission ON and Degas ON. When the device is configured for user filament selection:

1. The *Active Filament* attribute has Set access.
2. The *Active Degas Filament* attribute has Set access if the device supports individual selection of the Emission and Degas ON filaments. If the device uses the same filament for Emission and Degas ON, the *Active Degas Filament* has Get access.
3. If the user attempts to select a Filament that is broken, the device returns an “Invalid Attribute Value” error response.

For devices that support User Filament Selection Mode and which provide multiple filaments, the *Active Filament* and *Active Degas Filament* (if the device supports degas) attributes are required. The behavior of a device with more than one filament on is device specific. Generally, this will not be permitted and attempts to do so will return an error.

### 5-36.12.2.7 Sensor Warning

The Sensor Warning attribute provides 16 bits for the current Sensor Warning state of the object. The Sensor Warning attribute also maps to the S-Device Supervisor Exception Detail Warning attribute as specified in the Vacuum/Pressure Gauge device profile. The following table provides definitions for each Sensor Warning bit. Reserved bits must be set to zero.

**Table 5-36.31 Sensor Warning Attribute**

Data Component	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Sensor Warning Byte 0	Sensor Filament 8 Warning	Sensor Filament 7 Warning	Sensor Filament 6 Warning	Sensor Filament 5 Warning	Sensor Filament 4 Warning	Sensor Filament 3 Warning	Sensor Filament 2 Warning	Sensor Filament 1 Warning
Sensor Warning Byte 1	High Warning Exception	Low Warning Exception	Reserved	Sensor Warning *	Pressure Too High For Degas	Reserved	Electronics Warning	Exceeded Maximum C/E Ratio

\* Bit 4 of Byte 1, "Sensor Warning" relates to error conditions of the filament (cathode), anode, and ion collector.

### 5-36.12.2.8 Sensor Alarm

The Sensor Alarm attribute provides 16 bits for the current Sensor Alarm state of the object. The Sensor Alarm attribute also maps to the S-Device Supervisor Exception Detail Alarm attribute as specified in the Vacuum/Pressure Gauge device profile. The following table provides definitions for each Sensor Alarm bit. Reserved bits must be set to zero.

**Table 5-36.32 Sensor Alarm Attribute**

Data Component	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Sensor Alarm Byte 0	Sensor Filament 8 Alarm	Sensor Filament 7 Alarm	Sensor Filament 6 Alarm	Sensor Filament 5 Alarm	Sensor Filament 4 Alarm	Sensor Filament 3 Alarm	Sensor Filament 2 Alarm	Sensor Filament 1 Alarm
Sensor Alarm Byte 1	High Alarm Exception	Low Alarm Exception	Reserved	Environment Failure *	Overpressure Emission Off	Over Temperature of Electronics	Electronics Failure	Exceeded Maximum C/E Ratio

\* Bit 4 of Byte 1, "Environment Failure" relates to environment error conditions of the filament (cathode), anode, and ion collector which could cause an automatic emission off.

### 5-36.12.2.9 Status Extension

8 Bits providing the current sensor alarm state of the object.

**Table 5-36.33 Status Extension Attribute**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	Reserved	Reserved	Reserved	Reserved	Underrange Exceeded	Overrange Exceeded	Reading Invalid*

\* Note: Logical inversion of Reading Valid

### 5-36.12.3 Services

The following instance-level services are defined for this subclass of the S-Analog Sensor object:

**Table 5-36.34 Hot Cathode Ion Gauge Subclass Instance Attributes**

Service Code	Need in Implementation		Service Name	Description of Service
	Class	Instance		
61 <sub>hex</sub>	n/a	Optional	Set Degas State	Activate/deactivates degas mode according to the parameter Degas State. Degas mode may be terminated either automatically by device timeout, or remotely by this service.
62 <sub>hex</sub>	n/a	Conditional *	Set Emission State	Turns the filament on and off according to the parameter Emission State
63 <sub>hex</sub>	n/a	Conditional *	Clear Emission Off Alarm	Clears the Alarm attributes after an automatic "emission off". This service acknowledges and attempts to reset the Overpressure Emission Off and Exceeded Maximum C/E Ratio and Environment Failure Alarm attributes before an <i>Emission State ON</i> can be applied.

\* Required for single gauges only, otherwise support is optional

#### 5-36.12.3.1 Clear Emission Off Alarm Service

If the emission is switched off automatically in case of an overpressure or any other environmental (for example glow discharge) or Exceeded Maximum C/E Ratio condition, it is not possible to switch on the emission again before the related Alarm attribute is reset. This service resets the Alarm attributes: Overpressure Emission Off and Exceeded Maximum C/E Ratio and Environment Failure. See the State Event Matrix in section 5-36.11.2 for additional information.

#### 5-36.12.3.2 Set Emission State

**Table 5-36.35 Set Emission State Service Data Field Parameters**

Parameter	Required	Data Type	Description	Semantics of Values
EmissionState	Required	BOOL	See Behavior	0 = switches Emission OFF 1 = switches Emission ON

#### 5-36.12.3.3 Set Degas State

**Table 5-36.36 Set Degas State Service Data Field Parameters**

Parameter	Required	Data Type	Description	Semantics of Values
DegasState	Required	BOOL	See Behavior	0 = switches Degas OFF 1 = switches Degas ON

### 5-36.12.4 Behavior

#### 5-36.12.4.1 Sensor Alarms/Warnings: Filaments

When the *Mode Filament Selection* attribute is set to “User or Automatic Filament Selection”, if any bit of the *Sensor Alarm* attribute is set, the emission is switched off automatically. If a filament is broken, the corresponding bit in the *Sensor Warning* attribute is set.

When the *Mode Filament Selection* attribute is set to “**User Filament Selection**”, if the selected filament is broken, the corresponding bits in both the *Sensor Warning* and *Sensor Alarm* attributes are set and the device transitions to the state *Sensor Alarm Emission Off*. The alarm is cleared by selecting a new active filament and the device transitions to the state *Emission Off*. In this case the warning bit remains set and the alarm bit gets cleared.

When the *Mode Filament Selection* attribute is set to “**Automatic Filament Selection**”, the device sets all warning bits of all sensor filaments that are broken or failed. If the selected filament is broken, the device selects a new filament automatically and sets the warning bit of the new failed filament. If all filaments are broken, the corresponding bits in the *Sensor Alarm* attribute are set and the device transitions to the state *Sensor Alarm Emission Off*.

#### 5-36.12.4.2 Sensor Alarm Byte 1:

##### 5-36.12.4.2.1 Bit 0: Exceeded Maximum C/E Ratio

In case of an Exceeded Maximum C/E Ratio the device transitions to the state *Sensor Alarm Emission Off*. This error condition has to be acknowledged before the emission could be switched on again. This acknowledge is done by the service *Clear Emission Off Alarm*. After receiving this service, the device transitions to the state *Emission Off*.

##### 5-36.12.4.2.2 Bit 1: Electronics Failure

The device has detected an internal electronics failure which could not be cleared. The emission is switched off and the device transitions to the state *Sensor Alarm Emission Off*.

##### 5-36.12.4.2.3 Bit 2: Over Temperature of Electronics

The devices temperature is above its specified range. The emission is switched off and the device transitions to the state *Sensor Alarm Emission Off* and remains there until the temperature is below the critical limit. If the user attempts to switch on the emission while this bit is set, the error response Object State Conflict is returned. If the temperature goes below the critical limit, the device automatically transitions to the state *Emission Off*.

##### 5-36.12.4.2.4 Bit 3: Overpressure Emission Off

In case of an overpressure emission off, the user has to acknowledge the reason of the Emission Off situation. The acknowledge is done by the service: *Clear Emission Off Alarm*. After receiving this service, the device transitions to the state *Emission Off*.

##### 5-36.12.4.2.5 Bit 4: Environment Failure

Some other reasons than an overpressure condition (for example grid error or emission current being lost to a plasma) may result in an automatic emission switch off. This bit is set after any kind of these clearable emission switch off conditions. In this case the device transitions to the state *Sensor Alarm Emission Off*. The service *Clear Emission Off Alarm* is used to acknowledge the error condition and the device transitions to the state *Emission Off*.

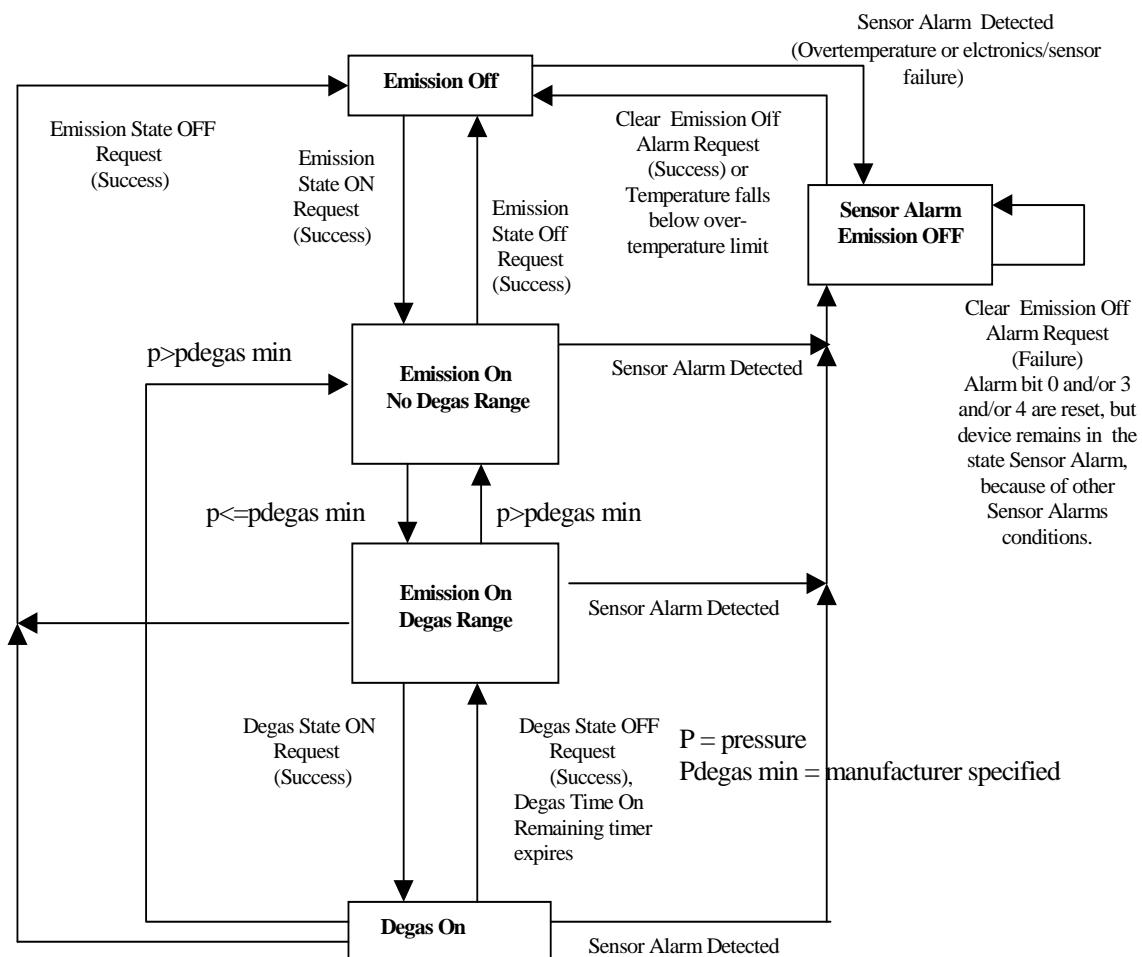
### 5-36.12.4.3 Sensor Warning Byte 1:

#### 5-36.12.4.3.1 Bit 3: Pressure Too High For Degas

If the pressure is above the manufacturer-specified range at which degas is allowed, this bit will be set. This bit is cleared if the pressure is below the manufacturer specified range at which degas is allowed.

### 5-36.12.5 S-Analog Sensor Object Sub-State for the Executing State

Figure 5-36.37 S-Analog Sensor Object Instance Behavior



See the S-Device Supervisor object (Section 5-35) for more information on the Executing State.

## 5-36.12.6 S-Analog Sensor Sub-State Event Matrix

Table 5-36.38 S-Analog Sensor Sub-State Event Matrix

Event	Sub-State				
	Emission OFF	Emission ON No Degas Range	Emission ON Degas Range	Sensor Alarm Emission Off	Degas On
Electronics Failure	Transition to <i>Sensor Alarm Emission Off</i>	Transition to <i>Sensor Alarm Emission Off</i>	Transition to <i>Sensor Alarm Emission Off</i>	Event reported in Sensor Alarm	Transition to <i>Sensor Alarm Emission Off</i>
Overtemperature	Transition to <i>Sensor Alarm Emission Off</i>	Transition to <i>Sensor Alarm Emission Off</i>	Transition to <i>Sensor Alarm Emission Off</i>	Event reported in Sensor Alarm	Transition to <i>Sensor Alarm Emission Off</i>
Temperature falls below overtemperature limit	Not Applicable	Not Applicable	Not Applicable	Transition to <i>Emission Off</i>	Not Applicable
Environmental Failure like glow discharge	Not Applicable	Transition to <i>Sensor Alarm Emission Off</i>	Transition to <i>Sensor Alarm Emission Off</i>	Not Applicable	Transition to <i>Sensor Alarm Emission Off</i>
Exceeded Maximum C/E Ratio	Not Applicable	Transition to <i>Sensor Alarm Emission Off</i>	Transition to <i>Sensor Alarm Emission Off</i>	Not Applicable	Transition to <i>Sensor Alarm Emission Off</i>
Clear Emission Off Alarm Request (Condition: <b>ONLY</b> Overpressure Emission Off and/or Sensor Failure and/or Exceeded Maximum C/E Ratio in Sensor Alarm set)	Not Applicable	Not Applicable	Not Applicable	Transition to <i>Emission OFF</i>	Not Applicable
Clear Emission Off Alarm Request (Condition: Overpressure Emission Off and/or Sensor Failure and/or Exceeded Maximum C/E Ratio <b>and</b> other Sensor Alarms set)	Not Applicable	Not Applicable	Not Applicable	<b>Error OSC **</b>	Not Applicable
Filament Alarm	Not Applicable	Transition to <i>Sensor Alarm Emission Off</i>	Transition to <i>Sensor Alarm Emission Off</i>	Not Applicable	Transition to <i>Sensor Alarm Emission Off</i>
Clear Emission Off Alarm Request (Sensor Alarms not set)	<b>Error AIRS *</b>	<b>Error OSC **</b>	<b>Error OSC **</b>	Not Applicable	<b>Error OSC **</b>
Set Emission State On Request and current pressure <= minimum degas pressure	Transition to <i>Emission ON Degas Range</i>	<b>Error AIRS *</b>	<b>Error AIRS *</b>	<b>Error OSC **</b>	<b>Error AIRS *</b>
Set Emission State On Request and current pressure > minimum degas pressure	Transition to <i>Emission ON No Degas Range</i>	<b>Error AIRS *</b>	<b>Error AIRS *</b>	<b>Error OSC **</b>	<b>Error AIRS *</b>
Set Emission State On Request at a broken filament in case of user filament selection	<b>Error OSC **</b>	<b>Error AIRS *</b>	<b>Error AIRS *</b>	<b>Error OSC **</b>	<b>Error OSC **</b>
Emission State On Request all filaments broken	Not Applicable	Not Applicable	Not Applicable	<b>Error OSC **</b>	Not Applicable

S-Analog Sensor Object, Class Code: 31<sub>Hex</sub>

Event	Sub-State				
	Emission OFF	Emission ON No Degas Range	Emission ON Degas Range	Sensor Alarm Emission Off	Degas On
Current Pressure <= minimum degas pressure	Ignore Event	Transition to <i>Emission ON Degas Range</i>	Ignore Event	Ignore Event	Ignore Event
Current Pressure <= minimum degas pressure	Ignore Event	Transition to <i>Emission ON Degas Range</i>	Ignore Event	Ignore Event	Ignore Event
Current Pressure > minimum degas pressure	Ignore Event	Ignore Event	Transition to <i>Emission ON No Degas Range</i>	Ignore Event	Transition to <i>Emission ON No Degas Range</i>
Valid Set Degas State On Request	Error OSC **	Error OSC **	Transition to <i>Degas On</i>	Error OSC **	Error AIRS *
Set Degas State On Request at a broken filament in case of user filament selection	Error OSC **	Error OSC **	Error OSC **	Error OSC **	Error AIRS *
Set Degas State On Request and <i>Degas Time Off Remaining</i> attribute > 0	Error OSC **	Error OSC **	Error OSC **	Error OSC **	Not Applicable
Set Degas State Off Request	Error OSC **	Error OSC **	Error OSC **	Error OSC **	Transition to <i>Emission ON Degas Range</i>
<i>Degas Time On Remaining</i> attribute decrements to zero	Not Applicable	Not Applicable	Not Applicable	Not Applicable	Transition to <i>Emission ON Degas Range</i>

\* Error AIRS = Error Response “Already in Requested Mode/State” (Code 0B<sub>hex</sub>)

\*\* Error OSC = Error Response “Object State Conflict” (Code 0C<sub>hex</sub>)

## 5-36.13 S-Analog Sensor Object Instance Subclass 06 - Transfer Function

The following specification applies to a subclass of this object.

### 5-36.13.1 Instance Attributes

**Table 5-36.39 Transfer Function Subclass Instance Attributes**

Attr ID	Need in Implem	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
96	Optional	Set	NV	Pressure Variable Mapping Function	USINT	Select function modifier of variable source to convert to selected units	0 = None (linear) 1 = Log() 2 = Exp() 3= Interpolate, manufacturer specific. 4-127 = Reserved by CIP 128-255 = Vendor Specific See Semantics section for usage [default] = 0

### 5-36.13.2 Semantics

#### 5-36.13.2.1 Pressure Variable Mapping Function

This attribute specifies an added function to the analog sensor object for the pressure variable sensor. This is to allow flexibility in connecting non-intelligent pressure devices to the Process Control Valve. The object uses:

$$\text{Value} = \text{Gain} (\text{Sensor Reading} + \text{Offset-A}) + \text{Offset-B}$$

Which is now embellished as:

$$\text{Value} = \text{Gain} (\text{Function}(\text{Sensor Reading} + \text{Offset-A})) + \text{Offset-B}$$

Where Function() is one of the following for each attribute value:

- 0: Value = Gain (Sensor Reading + Offset-A) + Offset-B
- 1: Value = Gain (Log10(Sensor Reading + Offset-A)) + Offset-B
- 2: Value = Gain (Exp10(Sensor Reading + Offset-A)) + Offset-B
- 3: Value = Gain (Interp(Sensor Reading + Offset-A)) + Offset-B

## 5-37 S-Analog Actuator Object

**Class Code: 32<sub>Hex</sub>**

The S-Analog Actuator Object models the interface to a physical actuator in a device. Associated with an analog actuator is a value which is corrected with an offset and a gain coefficient, optionally settable in the object before it is output to the physical actuator. Manufacturers may specify additional correction algorithms as extensions to this object.

The context of this object is that of an “Actuator” which should not be confused with a “Valve”. In particular, ZERO corresponds to the default powered-off state or typically NOT ACTUATED. For example, an S-Analog Actuator object physically connected to a *Normally Open Valve*, would result in the *Valve* being *ON* (i.e.: open or flowing) when the *Actuator* is *ZERO* (i.e.: not actuated).

Additionally, the S-Analog Actuator Object provides two sets of trip-point definitions. The behavior associated with these trip points is described in sections below.

This object is a member of the *Hierarchy of Semiconductor Equipment Devices*. As such, its behavior is managed by the S-Device Supervisor Object. See Section 5-35.

### 5-37.1 Class Attributes

**Table 5-37.1 S-Analog Actuator Object Class Attributes**

Attribute ID	Need in implementation	Access Rule	Name	Data Type	Description of Attribute
1 thru 7					These class attributes are optional and are described in Chapter 4 of this specification.
97-98					Reserved by CIP for Future Use
99	Conditional *	Get	Subclass	UINT	Identifies a subset of additional class attributes, services and behaviors.

\* If the value of Subclass is 00, which identifies "no subclass", then this attribute is OPTIONAL in implementation, otherwise, this attribute is REQUIRED.

### 5-37.1.1 Subclasses

Each class level subclass defines a unique meaning for an overlapping range of class attribute IDs and/or class service IDs. The range for subclass definitions begins at ID 96 and numbers downward for attributes, and ID 63<sub>hex</sub> and numbers downward for services. The subclass for a given class is identified by the value of its Subclass class attribute. The subclasses for this object are specified at the end of this object specification section.

## 5-37.2 Instance Attributes

Certain minimal implementations may support any optional “Set” attributes as “Get” only and still be compliant with this object specification. All required attributes must be supported as specified.

**Table 5-37.2 S-Analog Actuator Object Instance Attributes**

Attr ID	Need in Implementation	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
1	Optional	Get	NV	Number of Attributes	USINT	Number of supported attributes	The number of attributes supported by this object instance
2	Optional	Get	NV	Attribute List	ARRAY OF USINT	List of supported attribute	List of attributes supported by this object instance
3	Optional	See Semantics	NV	Data Type	USINT	Determines the Data Type of <i>Value</i> and all related attributes as specified in this table.	see Semantics section
4	Optional	See Semantics	NV	Data Units	ENGUNITS	Determines the context of <i>Value</i>	see Semantics section
5	Required	Set	V	Override	USINT	Specifies an override for the physical actuator. For values other than zero (normal control), the <i>Value</i> attribute is ignored.	0 = normal [default] see Semantics section
6	Required	Set	V	Value	INT or specified by <i>Data Type</i> if supported	Analog output value	The uncorrected value. see Semantics section [default] = 0
7	Required	Get	V	Status	BYTE	Alarm and Warning State of this object instance	see Semantics section [default] = 0
8	Optional	Set	NV	Alarm Enable	BOOL	Enables the setting of the Alarm Bit	0 = disable [default] 1 = enable
9	Optional	Set	NV	Warning Enable	BOOL	Enables the setting of the Warning Bit	0 = disable [default] 1 = enable
10	Optional	Set	NV	Offset	INT or specified by <i>Data Type</i> if supported	An amount to be added to <i>Value</i> prior to the application of gain	see Semantics section 0 = [default]

S-Analog Actuator Object, Class Code: 32<sub>Hex</sub>

Attr ID	Need in Implementation	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
11	Optional	Set	NV	Bias	INT or specified by <i>Data Type</i> if supported	An amount to be added to <i>Value</i> after the application of gain	see Semantics section 0 = [default]
12	Required if Attribute “Gain” is other than REAL	Get	NV	Gain Data Type	USINT	Determines the Data Type of attribute <i>Gain</i>	see Semantics section
13	Optional	Set	NV	Gain	REAL or specified by <i>Gain Data Type</i> if supported	An amount by which <i>Value</i> is scaled prior to driving the physical actuator	see Semantics section 1.0 = [default]
14	Required if Attribute 12 is other than REAL	Get	NV	Unity Gain Reference	REAL or specified by <i>Gain Data Type</i> if supported	Specifies the value of the <i>Gain</i> attribute equivalent to a gain of 1.0	Used for normalizing the <i>Gain</i> attribute. see Semantics section [default] = 1.0
15	Optional	Set	NV	Alarm Trip Point High	INT or specified by <i>Data Type</i> if supported	Determines the Value above which an Alarm Condition will occur	see Semantics section [default] = Maximum value for its data type.
16	Optional	Set	NV	Alarm Trip Point Low	INT or specified by <i>Data Type</i> if supported	Determines the Value below which an Alarm Condition will occur	see Semantics section [default] = Minimum value for its data type.
17	Optional	Set	NV	Alarm Hysteresis	INT or specified by <i>Data Type</i> if supported	Determines the amount by which the Value must recover to clear an Alarm Condition	see Semantics section [default] = 0
18	Optional	Set	NV	Warning Trip Point High	INT or specified by <i>Data Type</i> if supported	Determines the Value above which a Warning Condition will occur	see Semantics section [default] = Maximum value for its data type.
19	Optional	Set	NV	Warning Trip Point Low	INT or specified by <i>Data Type</i> if supported	Determines the Value below which a Warning Condition will occur	see Semantics section [default] = Minimum value for its data type.
20	Optional	Set	NV	Warning Hysteresis	INT or specified by <i>Data Type</i> if supported	Determines the amount by which the Value must recover to clear a Warning Condition	see Semantics section [default] = 0

**S-Analog Actuator Object, Class Code: 32<sub>Hex</sub>**

<b>Attr ID</b>	<b>Need in Implementation</b>	<b>Access Rule</b>	<b>NV</b>	<b>Name</b>	<b>Data Type</b>	<b>Description of Attribute</b>	<b>Semantics of Values</b>
21	Optional	Set	NV	Safe State	USINT	Specifies the behavior of the physical actuator for states other than Execute	see Semantics section 0 = [default]
22	Optional	Set	NV	Safe Value	INT or specified by <i>Data Type</i> if supported	The Value to be used for Safe State = Safe Value	see Semantics section 0 = [default]
97-98	Reserved by CIP for Future Use						
99	Conditional *	Get	NV	Subclass	UINT	Identifies a subset of additional instance attributes, services and behaviors.	0 = No subclass 1 – 65535 = Reserved

\* If the value of Subclass is 00, then this attribute is OPTIONAL in implementation, otherwise, this attribute is REQUIRED.

### 5-37.2.1 Subclasses

Each instance level subclass defines a unique meaning for an overlapping range of instance attribute IDs and/or instance service IDs. The range for subclass definitions begins at ID 96 and numbers downward for attributes, and ID 63<sub>hex</sub> and numbers downward for services. The subclass for a given instance is identified by the value of its Subclass instance attribute. The subclasses for this object are specified at the end of this object specification section.

### 5-37.2.2 Semantics:

#### 5-37.2.2.1 Data Type

All Data Type attributes, including *Data Type* and *Gain Data Type*, use the enumerated values specified in Appendix C-6.1.

The *Data Type* attribute is settable only in the *Idle State* and only if no attribute belonging to the object instance is the endpoint of an I/O connection in the *Established State*.

The *Data Type* attribute may change automatically based upon established I/O connections. See Behavior section for more information on this mechanism.

#### 5-37.2.2.2 Data Units

Specifies the context of *Value* and related attributes (such as, offset and trip points) for this object instance. See Appendix D for a list of values. A request to set attribute to an unsupported value will return an error response.

The *Data Units* attribute is settable only in the *Idle State*.

### 5-37.2.2.3 Value, Offset, Gain, Bias and Unity Gain Reference

The *Offset*, *Gain* and *Bias* attributes are applied to the *Value* attribute to derive the actual signal which drives the physical actuator. The gain is normalized using the *Unity Gain Reference* attribute value. (e.g., for an *UINT* type *Gain*, a *Unity Gain Reference* value may be 10000, allowing an effective gain of 0.0001 to 6.5535.)

The following formula applies:

$$\text{physical actuator drive signal} = \text{Gain}_N \bullet (\text{Value} + \text{Offset}) + \text{Bias}$$

where:  $\text{Gain}_N = \text{Gain} / \text{Unity Gain Reference}$

There may be additional nonlinear conversions applied to the drive signal as specified by the manufacturer.

### 5-37.2.2.4 Status

A bit mapped byte which indicates the Alarm and Warning Exception status of the object instance. The following definition applies:

**Table 5-37.3 Status Attribute**

Bit	Definition
0	High Alarm Exception: 0 = cleared; 1 = set
1	Low Alarm Exception: 0 = cleared; 1 = set
2	High Warning Exception: 0 = cleared; 1 = set
3	Low Warning Exception: 0 = cleared; 1 = set
4	Reserved
5	Reserved
6	Reserved
7	Reserved

### 5-37.2.2.5 Trip Points and Hysteresis

Trip Point High is the level above which the *Value* attribute will cause an Alarm or Warning exception condition.

Trip Point Low is the level below which the *Value* attribute will cause an Alarm or Warning exception condition.

A *Hysteresis* value specifies the amount by which the *Value* attribute must transition in order to clear an Alarm or Warning condition.

For example: A *Trip Point High* value of 90 and a *Hysteresis* value of 2 will result in an exception condition being set when the *Value* is above 90 and cleared when the *Value* drops below 88. Similarly, A *Trip Point Low* value of 90 and a *Hysteresis* value of 2 will result in an exception condition being set when the *Value* is below 90 and cleared when the *Value* increases above 92.

**5-37.2.2.6      Override**

This attribute is used to override the function of the *Value* attribute in driving the physical actuator. The primary application of this feature is in devices where the object instance is being driven by another object such as an S-Single Stage Controller object instance.

The *Safe State* attribute provides a mechanism for override depending upon object state and will take precedents over this. That is, if an object instance implements the *Safe State* attribute and related behavior, then this *Override* attribute and related behavior will only function in the Executing State.

**Table 5-37.4 Override Attribute**

Attribute Value	State
0	Normal
1	Zero
2	Maximum Value
3	Hold
4	Safe State
5-63	reserved
64-127	Device Specific
128-255	Vendor Specific

**5-37.2.2.7      Safe State**

This attribute specifies the behavior of the drive to the physical actuator for states other than Executing. See the S-Device Supervisor object definition in Section 5-35. for a description of object states. The following values are defined:

**Table 5-37.5 Safe State Attribute**

Attribute Value	State
0	Zero
1	Maximum Value
2	Hold Last Value
3	Use Safe Value
4-63	Reserved
64-127	Device Specific
128-255	Vendor Specific

**5-37.2.2.8      Safe Value**

For *Safe State* set to “Use Safe Value”, this attribute holds the value to which the actuator will be driven for object instance states other than Executing. Specifically, this attribute value will become the value of the *Value* attribute. Therefore, the correction formula specified above applies.

### 5-37.3 Common Services

The S-Analog Actuator Object provides the following Common Services:

**Table 5-37.6 S-Analog Actuator Object Common Services**

Service Code	Need in Implementation		Service Name	Description of Service
	Class	Instance		
0Ehex	Conditional*	Required	Get_Attribute_Single	Returns the contents of the specified attribute.
10hex	n/a	Required	Set_Attribute_Single	Modifies an attribute value.

\*The Get\_Attribute\_Single service is REQUIRED if any attributes are implemented.

See Appendix A for definition of these services

### 5-37.4 Object-Specific Services

The S-Analog Actuator Object provides no Object-Specific services.

### 5-37.5 Behavior

The behavior of this object is managed by the S-Device Supervisor Object in Section 5-35.6.

An S-Analog Actuator object instance modifies the *Value* by applying the formula specified above with the associated attribute values. *Value* is specified as *Data Type* and *Data Units*. Optionally, additional corrective algorithms are applied to further correct for various calibration effects. These additional algorithms are specified in other objects, as identified in the device profile, or as extensions, specified by the manufacturer.

All Trip Point calculations, as specified above, utilize the *Value* attribute before the application of *Offset* and *Gain*.

#### 5-37.5.1 Data Type

If the implementation of this object specifies more than one valid Data Type value, in the device profile or by vendor, then the following behavior with respect to *Data Type* applies: The Data Type value will be set automatically based upon the first valid I/O connection established by the device. This configuration will then remain in effect for this object instance even after all I/O connections are lost. If, however, a device is specified by a vendor to support only one Data Type, this behavior is not supported.

If no established I/O connections exist, which include an attribute from this object, then the *Data Type* attribute is settable provided that the object is in the *Idle State*.

The following example demonstrates this behavior:

A device profile specifies an instance of the S-Analog Actuator object as well as two static Assembly object instances, both with data attribute components mapped to this object instance. Assembly object instance ID 1 specifies INT data types and Assembly object instance ID 2 specifies REAL data types.

**S-Analog Actuator Object, Class Code: 32<sub>Hex</sub>**

After the device is On-Line, it is configured with an I/O connection to Assembly instance ID 2. When the connection transitions to the *Established State*, this object instance attribute *Data Type* is automatically set with the value for REAL before any data is communicated to, or from, the object instance.

## 5-38 S-Single Stage Controller Object

**Class Code: 33<sub>Hex</sub>**

The S-Single Stage Controller Object models a closed-loop control system within a device. Associated with a single stage controller is a Process Variable, a Setpoint and a Control Variable. As normally described by *classic control theory*, a closed-loop controller will drive the Control Variable in order to affect the value of the Process Variable such that it is made to equal the Setpoint. See the Semantics section, below, for more information regarding these variable definitions. Manufacturers may specify additional correction algorithms as extensions to this object.

This object is a member of the *Hierarchy of Semiconductor Equipment Devices*. As such, its behavior is managed by the S-Device Supervisor Object. See Section 5-35.

### 5-38.1 Class Attributes

**Table 5-38.1 S-Single Stage Controller Object Class Attributes**

Attribute ID	Need in implementation	Access Rule	Name	Data Type	Description of Attribute
1 thru 7	These class attributes are optional and are described in Chapter 4 of this specification.				
97-98	Reserved by CIP for Future Use				
99	Conditional *	Get	Subclass	UINT	Identifies a subset of additional class attributes, services and behaviors.

\* If the value of Subclass is 00, which identifies "no subclass", then this attribute is OPTIONAL in implementation, otherwise, this attribute is REQUIRED.

### 5-38.1.1 Subclasses

Each class level subclass defines a unique meaning for an overlapping range of class attribute IDs and/or class service IDs. The range for subclass definitions begins at ID 96 and numbers downward for attributes, and ID 63<sub>hex</sub> and numbers downward for services. The subclass for a given class is identified by the value of its Subclass class attribute. The subclasses for this object are specified at the end of this object specification section.

## 5-38.2 Instance Attributes

Certain minimal implementations may support any optional “Set” attributes as “Get” only and still be compliant with this object specification. All required attributes must be supported as specified.

**Table 5-38.2 S-Single Stage Controller Object Instance Attributes**

Attr ID	Need in Implementation	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
1	Optional	Get	NV	Number of Attributes	USINT	Number of supported attributes	Number of attributes supported in this object instance
2	Optional	Get	NV	Attribute List	ARRAY OF USINT	Attribute List	List of attributes supported in this object instance
3	Optional	See Semantics	NV	Data Type	USINT	Determines the Data Type of <i>Setpoint</i> , <i>Process Variable</i> and related attributes	see Semantics section
4	Optional	See Semantics	NV	Data Units	ENGUNITS	Determines the context of the Process related variables such as Setpoint and Process Variable	See Appendix D
5	Optional	Set	NV	Control Mode	USINT	Specifies the operational mode of the controller	See Semantic section [default] = Normal (0)
6	Required	Set	V	Setpoint	INT or specified by <i>Data Type</i> if supported	The setpoint to which the process variable will be controlled	See Semantics section. See Behavior section. 0 = [default]
7	Conditional *	Set	V	Process Variable	INT or specified by <i>Data Type</i> if supported	The measured process parameter	The device profile must specify the data connection for this attribute. It may be internally linked to a sensor. See Semantics section. 0 = [default]
8	Optional	Get	NV	CV Data Type	USINT	Determines the Data Type of <i>Control Variable</i>	see Semantics section
9	Conditional *	Get	V	Control Variable	INT or specified by <i>CV Data Type</i> if supported	The drive signal output of this object. The algorithm by which this attribute is calculated is manufacturer specific.	The device profile must specify the data connection for this attribute. It may be internally linked to an actuator. [default] = 0 See Semantics section.
10	Required	Get	V	Status	BYTE	Alarm and Warning State of this object instance	see Semantics section [default] = 0

S-Single Stage Controller Object, Class Code: 33<sub>Hex</sub>

Attr ID	Need in Implementation	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
11	Optional	Set	NV	Alarm Enable	BOOL	Enables the setting of the Alarm Status Bit	0 = disable [default] 1 = enable
12	Optional	Set	NV	Warning Enable	BOOL	Enables the setting of the Warning Status Bit	0 = disable [default] 1 = enable
13	Optional	Set	NV	Alarm Settling Time	UINT	Number of Milliseconds allowed for the control-loop to settle to within the error band	see Behavior section [default] = 0
14	Optional	Set	NV	Alarm Error Band	INT or specified by Data Type if supported	The amount by which the Setpoint must equal the Process Variable	see Behavior section [default] = 0
15	Optional	Set	NV	Warning Settling Time	UINT	Number of Milliseconds allowed for the control-loop to settle to within the Error Band	see Behavior section [default] = 0
16	Optional	Set	NV	Warning Error Band	INT or specified by Data Type if supported	The amount by which the Setpoint must equal the Process Variable	see Behavior section [default] = 0
17	Optional	Set	NV	Safe State	USINT	Specifies the Control Variable behavior for states other than Execute	see Semantics section 0 = [default]
18	Optional	Set	NV	Safe Value	INT or specified by Data Type if supported	The value to be used for Safe State = Safe Value	see Semantics section 0 = [default]
19	Optional	Set	NV	Ramp Rate	UDINT	Time in Milliseconds to reach Setpoint	0 = Disabled [default] x = value in milliseconds see Behavior section
85-96	Defined by subclasses below.						
97-98	Reserved by CIP for Future Use						
99	Conditional **	Get	NV	Subclass	UINT	Identifies a subset of additional instance attributes, services and behaviors.	0 = No subclass 1 = PID & Source Select 2 = DC Generator 3 = RF Generator 4 = Frequency Control 5 - 65535 = Reserved

\* The *Process Variable* is only optional if this device includes an internal sensor. Otherwise, the *Process Variable* is required. Similarly, The *Control Variable* is only optional if this device includes an internal actuator. Otherwise, the *Control Variable* is required.

\*\* If the value of Subclass is 00, then this attribute is OPTIONAL in implementation, otherwise, this attribute is REQUIRED.

### 5-38.2.1 Subclasses

Each instance level subclass defines a unique meaning for an overlapping range of instance attribute IDs and/or instance service IDs. The range for subclass definitions begins at ID 96 and numbers downward for attributes, and ID 63<sub>hex</sub> and numbers downward for services. The subclass for a given instance is identified by the value of its Subclass instance attribute. The subclasses for this object are specified at the end of this object specification section.

## 5-38.3 Semantics:

### 5-38.3.1 Data Type

All Data Type attributes, including *Data Type* and *CV Data Type*, use the enumerated values specified in Appendix C-6.1.

The *Data Type* attribute is settable only in the *Idle State* and only if no attribute belonging to the object instance is the endpoint of an I/O connection in the *Established State*.

The *Data Type* attribute may change automatically based upon established I/O connections. See Behavior section for more information on this mechanism.

### 5-38.3.2 Data Units

Specifies the context of *Setpoint* and *Process Variable* and related attributes (such as, offset and trip points) for this object instance. See Appendix D for a list of values. A request to set attribute to an unsupported value will return an error response.

The *Data Units* attribute is settable only in the *Idle State*.

In applications where this object is used in a relationship with an S-Analog Sensor object, this attribute may be specified as Get only, by the device profile or the vendor, where the value mirrors that of the S-Analog Sensor object *Data Units* attribute.

### 5-38.3.3 Setpoint, Process Variable and Control Variable

These three attributes compose the primary aspects of basic closed-loop control. The *Process Variable* is the measured parameter of the process or system being controlled. The *Setpoint* is the desired value for the measured parameter. By affecting the value of the *Control Variable*, the closed-loop controller drives the process or system to the desired state of:

Process Variable = Setpoint

The *Control Variable* is, therefore, connected to the process or system in such a way that it affects the value of the *Process Variable*. Examples of *Control Variable / Process Variable* combinations include: heater / temperature; valve / flow; or regulator / pressure.

### 5-38.3.4 Status

A bit mapped byte which indicates the Alarm and Warning Exception status of the object instance. The following definition applies:

**Table 5-38.3 Status Attribute**

Bit	Definition
0	Alarm Exception: 0 = cleared; 1 = set
1	Warning Exception: 0 = cleared; 1 = set
2	Reserved
3	Reserved
4	Reserved
5	Reserved
6	Reserved
7	Reserved

### 5-38.3.5 Control Mode

This attribute is used to override the value of the *Control Variable* attribute. Further, it may cause the object to modify the internal control algorithm such that a smooth, or “bumpless” transitions occurs upon activating control to setpoint.

The *Safe State* attribute provides a mechanism for override depending upon object state and will take precedents over this. That is, if an object instance implements the *Safe State* attribute and related behavior, then this *Override* attribute and related behavior will only function in the Executing State.

**Table 5-38.4 Control Mode Attribute**

Attribute Value	State
0	Normal
1	Zero / Off / Closed
2	Full / On / Open
3	Hold
4	Safe State
5-63	reserved
64-127	Device Specific (specified by device profile)
128-255	Vendor Specific

### 5-38.3.6 Safe State

This attribute specifies what value will be held in the *Control Variable* attribute for states other than Executing. See the S-Device Supervisor object definition in Section 5-35. for a description of object states. The following values are defined:

**Table 5-38.5 Safe State Attribute**

Attribute Value	State
0	Zero / Off
1	Full Scale / On
2	Hold Last Value
3	Use Safe Value
4-63	reserved
64-127	Device Specific (specified by device profile)
128-255	Vendor Specific

**5-38.3.7 Safe Value**

For Safe State set to Use Safe Value, this attribute holds the value to which the Control Variable attribute will be set for object instance states other than Executing.

**5-38.4 Common Services**

The S-Single Stage Controller Object provides the following Common Services:

**Table 5-38.6 S-Single Stage Controller Object Common Services**

Service Code	Need in Implementation		Service Name	Description of Service
	Class	Instance		
0E <sub>hex</sub>	Conditional*	Required	Get_Attribute_Single	Returns the contents of the specified attribute.
10 <sub>hex</sub>	n/a	Required	Set_Attribute_Single	Modifies an attribute value.

\*The Get\_Attribute\_Single service is REQUIRED if any attributes are implemented.

See Appendix A for definitions of these services

**5-38.5 Object-specific Services**

The S-Single Stage Controller Object provides no Object-Specific services.

**5-38.6 Behavior**

The behavior of this object is managed by the S-Device Supervisor Object in Section 5-35.5. Additionally, this object exhibits the following behavior:

**5-38.6.1 Alarm and Warning Exception Conditions**

While in the Executing State as defined by the S-Device Supervisor Object: Immediately upon detecting that the Setpoint does not equal the Process Variable by an amount plus-or-minus the associated (alarm or warning) Error Band, a timer is started. This internal timer is incremented as long as the above condition exists. If the timer exceeds the amount indicated by the associated (alarm or warning) Settling Time and the associated (alarm or warning) Exception Enable is set, then the appropriate (alarm or warning) Exception Condition is set. Note that two internal timers are required in order to support both Alarm and Warning Exception reporting.

This behavior is modified for Ramp Rate values not equal to zero. In such cases, the timer is not enabled until after the expiration of the Ramp Time (see Behavior description below).

#### **5-38.6.2 Ramp Rate**

For Ramp Rate values other than zero, the S-Single Stage Controller Object internally modifies the Setpoint value in such a way that the Process Variable is “ramped” to its final value. For example: with a Ramp Rate value of 1000, a new Setpoint value will be internally (transparently) modified, in whatever time increments the object is able to sustain, in order to affect a smooth transition over one second from the old Setpoint to the new Setpoint, finally reaching the new Setpoint at the one second mark.

#### **5-38.6.3 Data Type**

If the implementation of this object specifies more than one valid Data Type value, in the device profile or by vendor, then the following behavior with respect to *Data Type* applies: The Data Type value will be set automatically based upon the first valid I/O connection established by the device. This configuration will then remain in effect for this object instance even after all I/O connections are lost. If, however, a device is specified by a vendor to support only one Data Type, this behavior is not supported.

If no established I/O connections exist, which include an attribute from this object, then the *Data Type* attribute is settable provided that the object is in the *Idle State*.

The following example demonstrates this behavior:

A device profile specifies an instance of the S-Single Stage Controller object as well as two static Assembly object instances, both with data attribute components mapped to this object instance. Assembly object instance ID 1 specifies INT data types and Assembly object instance ID 2 specifies REAL data types.

After the device is On-Line, it is configured with an I/O connection to Assembly instance ID 2. When the connection transitions to the *Established State*, this object instance attribute *Data Type* is automatically set with the value for REAL before any data is communicated to, or from, the object instance.

#### **5-38.6.4 Control**

The application of this object is further specified in the applicable device profile; primarily, the interfaces and object relationships are defined. Generally, the *Process Variable* attribute is restricted to "Get Only" access and an internal connection is defined to another object. Similarly, the *Control Variable* is generally not supported due to internal connections.

When in the EXECUTING state, this object is running an application process designed to cause the *Process Variable* to be driven to the value of the *Setpoint*. In any state other than EXECUTING, the application process is stopped and the *Safe State* is activated for the output of the object.

Any fault detected by the object application process causes the object to transition to the appropriate state as defined by the managing S-Device Supervisor object.

## 5-38.7 S-Single Stage Controller Object Instance Subclass 01 - (PID and Source Select)

The following specification applies to an instance subclass of this object. The intended applications of this subclass include cases where: PID (Proportion, Integral, Derivative) Servo Control algorithms are utilized and/or more than one Process Variable Source is connected. A typical application for the object instance subclass is that of a Pressure Control Valve.

### 5-38.7.1 Instance Attributes

The following Instance Attributes are specific for this subclass.

**Table 5-38.7 PID and Source Select Subclass Instance Attributes**

Attr ID	Need in Implementation	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
87	Conditional *	Get	V	Calibrating State	BOOL	Indicates whether or not the object is in the CALIBRATING state	0 = Not CALIBRATING 1 = CALIBRATING [default] = 0
88	Optional	Set	NV	Delay Time	UINT	Defines the time necessary for crossover condition to be present before switching ranges	Time in milliseconds [default] = manufacturer specific see Behavior section
89	Conditional **	Set	NV	Sensor Crossover High	INT or specified by Data Type if supported	Defines the crossover point between High to Low range inputs.	[default] = manufacturer specific see Behavior section see Semantics section
90	Conditional **	Set	NV	Sensor Crossover Low	INT or specified by Data Type if supported	Defines the crossover point between Low to High range inputs.	[default] = manufacturer specific see Behavior section see Semantics section
91	Optional	Set	NV	Expected Process Parameter	INT or specified by Data Type is supported	Used to optimize control function	[default] = manufacturer specific
92	Optional	Set	NV	Kd	REAL	Derivative Gain	[default] = manufacturer specific
93	Optional	Set	NV	Ki	REAL	Integral Gain	[default] = manufacturer specific
94	Optional	Set	NV	Kp	REAL	Proportional Gain	[default] = manufacturer specific
95	Optional	Set	NV	Control Direction	BOOL	Defines control direction as Direct or Reverse	0 = Direct (Downstream Control) 1 = Reverse (Upstream Control) [default] = 0 see Behavior section

**S-Single Stage Controller Object, Class Code: 33<sub>Hex</sub>**

Attr ID	Need in Implementation	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
96	Optional	Set	NV	Process Variable Source	USINT	Source of the Process Variable value for the process control function	0 = Network 1 = S-Analog Sensor object instance #1 2 = S-Analog Sensor object instance #2 3 = Auto 4 = Disable 5-255 = Reserved by CIP [default] = 1 see Behavior section

\* Required if Calibrating service is supported, otherwise, not required.

\*\* Required if Process Variable Source type 3 is supported, otherwise, not required.

## 5-38.7.2 Semantics

### 5-38.7.2.1 Calibrating State

Indicates the state of the object instance.

### 5-38.7.2.2 Delay Time

Specifies the required time of the presence of the Sensor Crossover condition to be true before changing the *Process Variable* source. See Behavior section for more information.

### 5-38.7.2.3 Sensor Crossover High, Low

Specifies the value of the *Process Variable*, in percent, that allows automatic switching to the other *Process Variable Sources*. See Behavior section for more information.

### 5-38.7.2.4 Expected Process Parameter

Specifies a value for a process parameter that may be applied to the control application process to optimize the control of the *Process Variable*. The context of this attribute is defined by the application. For example, a Pressure Control Valve application may define this attribute to specify the expected Flow Rate through the valve.

### 5-38.7.2.5 Kd,Ki,Kp

Specifies the values of a PID control loop. This gives access to adjusting the parameters of the control function for the *Process Variable*.

### 5-38.7.2.6 Control Direction

This defines the polarity for control of the process variable. The control mechanism can then be knowledgeable of what effect increasing or decreasing the *Control Variable* will have on the *Process Variable*. See Behavior section for more information.

### 5-38.7.2.7 Process Variable Source

The *Process Variable Source* specifies the source of the measured value, or the process variable, being controlled. See Behavior section for more information.

**S-Single Stage Controller Object, Class Code: 33<sub>Hex</sub>**

If the Process Variable Source equals zero indicating a Network source, the value of the Process Variable attribute will be updated across a network connection. Sensor Crossover parameters are not used, nor may any automatic switching of the Process Variable Source occur.

If the Process Variable Source equals one or two, the value of the Process Variable attribute is fed by the corresponding S-Analog Sensor object Value attribute. Sensor Crossover parameters are not used nor, may any automatic switching of the Process Variable Source occur.

If the Process Variable Source equals three, the value of the Process Variable attribute is fed by either the low-range or the high-range S-Analog Sensor object Value attribute. In order for three to be a valid attribute value for the Process Variable Source, two S-Analog Sensor instances are required. Selection of the appropriate S-Analog Sensor instance is performed by the device, and defined by the Sensor Crossover High and Sensor Crossover Low attribute values defined below.

Instance 1 for the Process Variable Source S-Analog Sensor is taken as the low range and also is the required instance for direct input when only one Process Variable Source is attached to the device. Instance 2 is taken as the high range, defining this as a higher magnitude above the low range.

**NOTE:** The capability of each device to accurately resolve a particular high range or low range sensors must be specified by the manufacturer. Inappropriate selection of high and low range sensors needs to be addressed in application guides for usage of the device. Care must be given to the compatibility of these ranges to avoid unsatisfactory behavior.

A value of 4 for the Process Variable Source supports device configurations where no process variable exists for this instance. This value prevents action by the controller until configured with a valid Process Variable Source.

### 5-38.7.3 Services

The following Services are specific for this subclass.

**Table 5-38.8 PID and Source Select Subclass Services**

Service Code	Need in implementation		Service Name	Description of Service
	Class	Instance		
63 <sub>hex</sub>	n/a	Optional	Calibrate	Moves the device to the <b>Calibrating</b> state. This service may also end a calibration that is in process.

### 5-38.7.3.1 Calibrate

Used to transition this object to the CALIBRATING substate of the EXECUTING state. Refer to the Behavior section for a description of the CALIBRATING state. This service may also end a calibration procedure that is in process.

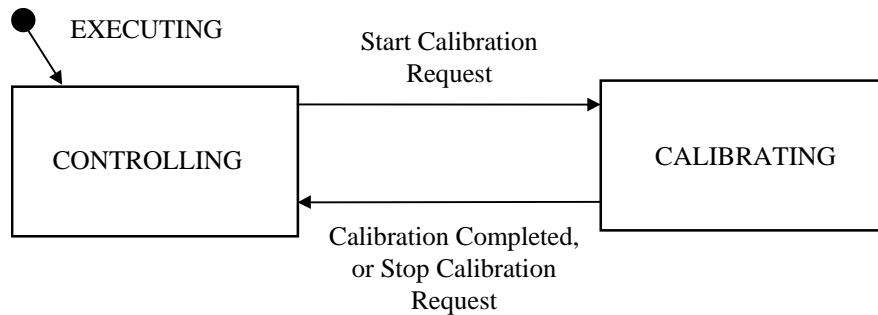
**Table 5-38.9 Calibrate Request Service Data Field Parameters**

Parameter	Data Type	Description	Semantics of Values
Command	USINT	Specifies whether to start or stop the calibration	0 = Stop Calibration 1 = Start Calibration

### 5-38.7.4 Behavior

The following additional behavior is specific for this object instance subclass.

**Figure 5-38.10 Object State Transition Diagram**



**Table 5-38.11 State Descriptions**

State	Description
CONTROLLING	The object is running an application process designed to position the control valve such that the <i>Process Variable</i> is driven to equal the <i>Setpoint</i> .
CALIBRATING	The object is running a manufacturer-specific calibration process. This process may involve an algorithm designed to characterize the system in order to improve the quality of control.

**Table 5-38.12 Object State Event Matrix**

EVENT	STATE	
	Controlling	Calibrating
Start Calibration Request	Transition to CALIBRATING	Error AIRS*
Stop Calibration Request	Error AIRS*	Transition to CONTROLLING
Calibration Complete	—	Transition to CONTROLLING

\* Error AIRS = Error Response - Already in Requested Mode/State

#### 5-38.7.4.1 Process Variable Input Switching

The Sensor Crossover High/Low Variable values for each Process Variable Source defines the value to change to the other Process Variable source. This is only valid with the Process Variable Source Attribute set to automatic (3), which can only be valid with the presence of two Process instances.

#### 5-38.7.4.2 Delay Time

The Delay Time timer is used to minimize unnecessary switching of the Process Variable Source when more than one source is available and automatic switching is selected. The Delay Time shall be held in reset when the condition for Sensor Crossover is false, and starts when the condition is true. After starting to count, the Delay Time resets anytime the condition is not true. The Delay Time also resets after a switch has occurred.

#### 5-38.7.4.3 Sensor Crossover Low

This parameter is only used when there are two instances of the S-Analog Object for Process Variable Sources and the value of the Process Variable Source is 3. It specifies the value of the Process Variable that allows the device to switch automatically from the Process Variable Source instance 1 to instance 2. Its value is valid only in the range of the instance 1 S-Analog Sensor. To rephrase this, this variable describes a value when to automatically switch from the low to high range of Process Variable Sources. Specifying a value outside the range of the low range sensor is not allowed.

#### 5-38.7.4.4 Sensor Crossover High

This parameter only used when there are two instances of the S-Analog Object for Process Variable Sources and the value of the Process Variable Source is 3. It specifies the value of the Process Variable that allows automatic switching from the Process Variable Source instance 2 to instance 1. Its value is valid only in the range of the instance 2 S-Analog Sensor. To rephrase this, this variable describes a value when to automatically switch from the high to low range of Process Variable Sources. Specifying a value outside the range of the high range sensor is not allowed.

#### 5-38.7.4.5 Control Direction

*Direct-Acting Controller* — as the *Process Variable* increases, the *Control Variable* increases (and visa versa). For example, where this object instance subclass is applied to a variable conductance valve downstream of a pressure control process (downstream pressure control); an increase in pressure (*Process Variable*) causes an increase in valve opening (*Control Variable*) resulting in a commensurate decrease in pressure, whereby, the pressure is maintained at *Setpoint*.

*Reverse-Acting Controller* — as the *Process Variable* increases, *Control Variable* decreases (and visa versa). Referencing the above example (*Direct-Acting Controller*), this configuration would apply to a variable conductance valve upstream of a pressure control process (upstream pressure control).

#### **5-38.7.4.6    Process Variable Source**

If the *Process Variable Source* equals zero indicating a Network source, the value of the *Process Variable* attribute will be updated across a network connection. Sensor Crossover parameters are not used, nor may any automatic switching of the Process Variable Source occur.

If the *Process Variable Source* equals one or two, the value of the *Process Variable* attribute is fed by the corresponding S-Analog Sensor object *Value* attribute. Sensor Crossover parameters are not used nor, may any automatic switching of the Process Variable Source occur.

If the *Process Variable Source* equals three, the value of the *Process Variable* attribute is fed by either the low-range or the high-range S-Analog Sensor object *Value* attribute. In order for three to be a valid attribute value for the *Process Variable Source*, two S-Analog Sensor instances are required. Selection of the appropriate S-Analog Sensor instance is performed by the device, and defined by the *Sensor Crossover High* and *Sensor Crossover Low* attribute values defined below.

Instance 1 for the *Process Variable Source* S-Analog Sensor is taken as the low range and also is the required instance for direct input when only one *Process Variable Source* is attached to the device. Instance 2 is taken as the high range, defining this as a higher magnitude above the low range.

**NOTE:** The capability of each device to accurately resolve a particular high range or low range sensors must be specified by the manufacturer. Inappropriate selection of high and low range sensors needs to be addressed in application guides for usage of the device. Care must be given to the compatibility of these ranges to avoid unsatisfactory behavior.

A value of 4 for the *Process Variable Source* supports device configurations where no process variable exists for this instance. This value prevents action by the controller until configured with a valid *Process Variable Source*.

## 5-38.8 S-Single Stage Controller Instance Subclass 02 - DC Generator

The following specification applies to a subclass of this object for application in DC Power Generator devices such as those used to create plasma energy.

### 5-38.8.1 Instance Attributes

The following Instance Attributes are specified for this object subclass.

**Table 5-38.13 DC Generator Subclass Instance Attributes**

Attr ID	Need in Implementation	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
85	Optional	Get	NV	Output Max	DINT	Factory set Device Capability see Below	Watts, Volts or milliAmps as Specified by <i>Data Units</i> attribute (ID 4)
86	Optional	Set	NV	Output Limit	DINT	Desired limit of the Device see Below	Watts, Volts or milliAmps as Specified by <i>Data Units</i> attribute (ID 4)
87	Optional	Set	NV	Arc Trip Level	DINT	Output voltage level which determines an arc condition	Volts
88	Optional	Set	NV	Arc Count Limit	DINT	Number of arcs that create an Arc Limit Exceeded Warning Condition	Counts [default] = 1
89	Optional	Get	V	Arc Counter	UDINT	Number of arcs detected since Output Power was last started	Counts
90	Conditional*	Set	NV	Pulse Enable	BOOL	Enables the Pulse Application Process See Below	0 = Disable [default] 1 = Enable
91	Conditional*	Set	NV	Pulse Second Setpoint	INT or specified by <i>Data Type</i> if supported	The setpoint value for the second half of a pulse period	Watts, Volts or milliAmps as Specified by <i>Data Units</i> attribute (ID 4) [default] = 0
92	Conditional*	Set	NV	Pulse Period Duration	DINT	The duration of one complete on/off cycle	microseconds
93	Conditional*	Set	NV	Pulse Duty Cycle	UINT	The percentage of the Pulse Period for which the setpoint is at its normal value	%
94	Optional	Set	NV	Sync Enable	BOOL	Enables an external synchronizing signal to begin a Pulse Period	0 = Disabled [default] 1 = Enabled
95	Optional	Set	NV	Sync Phase Delay	DINT	Specifies a delay between the sync signal and beginning of the Pulse Period	microseconds

**S-Single Stage Controller Object, Class Code: 33<sub>Hex</sub>**

<b>Attr ID</b>	<b>Need in Implementation</b>	<b>Access Rule</b>	<b>NV</b>	<b>Name</b>	<b>Data Type</b>	<b>Description of Attribute</b>	<b>Semantics of Values</b>
96	Optional	Set	NV	Ramp Rate Increment	UINT	Specifies a setpoint ramp as described below	Range = 0 to 4095 0 - 100% Setpoint per Second 0 = Disabled [default]

\* Required if Pulsed Power Delivery is supported

**5-38.8.2 Semantics****5-38.8.2.1 Output Max**

This is a factory set specification of the capability of the device. The context of the attribute is determined by the value of the *Data Units* attribute (ID 4). Depending upon the use of this Object Instance, this value specifies the maximum, Output Power, Output Voltage or Output Current.

**5-38.8.2.2 Output Limit**

This attribute is set by the user to limit the output. The device shall not deliver any output beyond this value. The context of the attribute is the same as that of the *Output Max* attribute above. A request to set this attribute beyond the value of *Output Max* returns an Invalid Attribute Value error.

**5-38.8.2.3 Ramp Rate Increment**

Upon receiving a new setpoint value for output power, the S-Single Stage Controller Object instance will internally ramp to the new setpoint value as specified by the *Ramp Rate Increment* attribute. For a *Ramp Rate Increment* value of zero, the ramping is disabled and the new setpoint takes effect immediately or as determined by the time based *Ramp Rate* attribute (ID 19) value. Otherwise, the *Ramp Rate Increment* behavior overrides that of the time based *Ramp Rate*.

For values in the range of 1-4095, corresponding to the range of  $2.442 \times 10^{-4}$  % to 100 %, the setpoint is ramped at a rate equal to the percentage of the difference between the initial setpoint and the new setpoint per second (e.g., 50% specifies half the difference).

For example: Ramp Rate = 2047 (50%); initial setpoint = 75% of Full Scale; new setpoint = 100% of Full Scale. Results: effective setpoint ramps from 75% to 100% of Full Scale at the rate of 12.5% of Full Scale per second and reaching final value in 2 seconds.

**5-38.8.3 Services**

There are no additional services defined for this subclass

#### **5-38.8.4 Behavior**

##### **5-38.8.4.1 Pulse Application Process**

The Pulse Application Process causes the S-Single Stage Controller Object Instance to behave as though its setpoint attribute is periodically switching between two values. The first value is that which is set to the Setpoint attribute. The second value is that which is set to the Pulse Second Setpoint attribute.

## 5-38.9 S-Single Stage Controller Instance Subclass 03 - RF Generator

The following specification applies to a subclass of this object for application in RF Power Generator devices such as those used to create plasma energy.

### 5-38.9.1 Instance Attributes

The following Instance Attributes are specified for this object subclass.

**Table 5-38.14 RF Generator Subclass Instance Attributes**

Attr ID	Need in Implementation	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
88	Optional	Get	NV	Output Max	DINT	Factory set Device Capability see Below	Watts, milliWatts, Volts or milliAmps as Specified by <i>Data Units</i> attribute (ID 4)
89	Optional	Set	NV	Output Limit	DINT	Desired limit of the Device see Below	Watts, milliWatts, Volts or milliAmps as Specified by <i>Data Units</i> attribute (ID 4)
90	Conditional*	Set	NV	Pulse Enable	BOOL	Enables the Pulse Application Process	0 = Disable [default] 1 = Enable
91	Conditional*	Set	NV	Pulse Second Setpoint	DINT	The setpoint value for the second half of a pulse period	Watts, milliWatts, Volts or milliAmps as Specified by <i>Data Units</i> attribute (ID 4) [default] = 0
92	Conditional*	Set	NV	Pulse Period Duration	DINT	The duration of one complete on/off cycle	microseconds
93	Conditional*	Set	NV	Pulse Duty Cycle	UINT	The percentage of the Pulse Period for which the setpoint is at its normal value	%
94	Optional	Set	NV	Sync Enable	BOOL	Enables an external synchronizing signal to begin a Pulse Period	0 = Disabled [default] 1 = Enabled
95	Optional	Set	NV	Sync Phase Delay	DINT	Specifies a delay between the sync signal and beginning of the Pulse Period	microseconds
96	Optional	Set	NV	Ramp Rate Increment	UINT	Specifies a setpoint ramp as described below	Range = 0 to 4095 0 - 100% Setpoint per Second 0 = Disabled [default]

\* Required if Energy Control is supported

\*\* Required if Target Life tracking is supported

\*\*\* Required if Pulsed Power Delivery is supported

**5-38.9.2 Semantics****5-38.9.2.1 Output Max**

This is a Factory set specification of the capability of the device. The context of the attribute is determined by the value of the *Data Units* attribute (ID 4). Depending upon the use of this Object Instance, this value specifies the maximum, Output Power or Delivered Power.

**5-38.9.2.2 Output Limit**

This attribute sets the limit beyond which the device will not operate. The context of the attribute is the same as that of the *Output Max* attribute above. A request to set this attribute beyond the value of *Output Max* returns an Invalid Attribute Value error.

**5-38.9.2.3 Ramp Rate Increment**

Upon receiving a new setpoint value for output power, the S-Single Stage Controller Object instance will internally ramp to the new setpoint value as specified by the *Ramp Rate Increment* attribute. For a *Ramp Rate Increment* value of zero, the ramping is disabled and the new setpoint takes effect immediately or as determined by the time based *Ramp Rate* attribute (ID 19) value. Otherwise, the Ramp Rate Increment behavior overrides that of the time based Ramp Rate.

For values in the range of 1-4095, corresponding to the range of  $2.442 \times 10^{-4}$  % to 100 %, the setpoint is ramped at a rate equal to the percentage of the difference between the initial setpoint and the new setpoint per second (e.g., 50% specifies half the difference).

For example: Ramp Rate = 2047 (50%); initial setpoint = 75% of Full Scale; new setpoint = 100% of Full Scale. Results: effective setpoint ramps from 75% to 100% of Full Scale at the rate of 12.5% of Full Scale per second and reaching final value in 2 seconds.

**5-38.9.3 Services**

There are no additional services defined for this subclass.

**5-38.9.4 Behavior****5-38.9.4.1 Pulse Application Process**

The Pulse Application Process causes the S-Single Stage Controller Object Instance to behave as though its setpoint attribute is periodically switching between two values. The first value is that which is set to the Setpoint attribute. The second value is that which is set to the Pulse Second Setpoint attribute.

## 5-38.10 S-Single Stage Controller Instance Subclass 04 - Frequency Control

The following specification applies to a subclass of this object for application in RF Power Generator devices such as those used to create plasma energy.

### 5-38.10.1 Instance Attributes

The following Instance Attributes are specified for this object subclass.

**Table 5-38.15 Frequency Control Subclass Instance Attributes**

Attr ID	Need in Implementation	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
92	Optional	Get	NV	Minimum Frequency	DINT	Factory Set	Hertz
93	Optional	Get	NV	Maximum Frequency	DINT	Factory Set	Hertz
94	Optional	Set	NV	Default Frequency	DINT	Factory Set	Hertz
95	Optional	Set	NV	Frequency Limit Low	DINT	Limits the Output Frequency of the device	Hertz May not be set beyond Min Frequency
96	Optional	Set	NV	Frequency Limit High	DINT	Limits the Output Frequency of the device	Hertz May not be set beyond Max Frequency

### 5-38.10.2 Semantics

#### 5-38.10.2.1 Minimum, Maximum and Default Frequency Attributes

*Minimum, Maximum and Default Frequency* attributes specify the capabilities of the device.

#### 5-38.10.2.2 Frequency Limit High and Low Attributes

*Limit* attributes allow the user to set bounds beyond which the device will not operate. A request to set a *limit* attribute or the *default* attribute beyond the bounds of a specified *Min* or *Max* value will return an Invalid Attribute Value error.

#### 5-38.10.2.3 Setpoint

A Setpoint value of zero causes the S-Single Stage Controller object instance to perform the Auto-tune Application Process. Any other value of setpoint (within specified limits) causes the S-Single Stage Controller object instance to perform the Frequency Control Application Process.

*Data Type* = DINT

*Data Units* = Hertz

### **5-38.10.3 Services**

There are no additional services defined for this subclass.

### **5-38.10.4 Behavior**

#### **5-38.10.4.1 Frequency Control Application Process**

This application process controls the output frequency to the set value as specified by the Setpoint attribute of this object instance.

#### **5-38.10.4.2 Auto-tune Application Process**

This application process controls the output frequency in order to affect optimum power delivery efficiency in RF Power Generator devices. The vendor specific algorithm tunes the output frequency in order to minimize Reflected Power for the device's application.

## **5-39      S-Gas Calibration Object**

### **Class Code: 34Hex**

An S-Gas Calibration Object affects the behavior of an associated S-Analog Sensor object instance; a device profile will show a relationship between these two objects where an S-Gas Calibration Object is used. The S-Analog Sensor object uses a selection attribute as the gas type selection mechanism. The S-Gas Calibration Object provides the data with which a device enacts the appropriate calibration algorithm for a given gas type. Each S-Gas Calibration Object Instance contains a set of attribute values for one particular calibration set; each identified by the Gas Standard Number.

The S-Gas Calibration class level object provides a service for retrieving a list of all valid object instances. The service response includes a list of elements. Each element includes: Instance ID, Gas Standard Number and the valid S-Analog Sensor object instance ID for which the instance is valid.

There may be more than one instance with the same Gas Standard Number. These instances may be differentiated by Full Scale, Gas Symbol, Additional Scaler and/or other parametric distinctions, including valid S-Analog Sensor object instance ID. The distinctions may, or may not, be evident in the Get\_All\_Instances service response, depending upon what the distinction is.

S-Gas Calibration Objects most often utilize the region of Manufacturer Specified Attributes (ID > 100) for specific calibration parameters.

This object is a member of the *Hierarchy of Semiconductor Equipment Devices*. As such, its behavior is managed by the Device Supervisor Object. See Section 5-35.

The S-Gas Calibration object makes use of a list of Standard Gas Type Numbers. This list is described in publication:

SEMI E52-95 “Practice for Referencing Gases Used in Digital Mass Flow Controllers”, Semiconductor Equipment and Materials International (SEMI), Mountain View, CA 94043-4080.

NOTE: It is implied that the reference above is to the latest revision as specified by SEMI.

## 5-39.1 Class Attributes

**Table 5-39.1 S-Gas Calibration Object Class Attributes**

Attribute ID	Need in implementation	Access Rule	Name	Data Type	Description of Attribute
1 thru 7	These class attributes are optional and are described in Chapter 4 of this specification.				
97-98	Reserved by CIP for Future Use				
99	Conditional *	Get	Subclass	UINT	Identifies a subset of additional class attributes, services and behaviors.

\* If the value of Subclass is 00, which identifies "no subclass", then this attribute is OPTIONAL in implementation, otherwise, this attribute is REQUIRED.

### 5-39.1.1 Subclasses

Each class level subclass defines a unique meaning for an overlapping range of class attribute IDs and/or class service IDs. The range for subclass definitions begins at ID 96 and numbers downward for attributes, and ID 63<sub>hex</sub> and numbers downward for services. The subclass for a given class is identified by the value of its Subclass class attribute. The subclasses for this object are specified at the end of this object specification section.

## 5-39.2 Instance Attributes

Certain minimal implementations may support any optional "Set" attributes as "Get" only and still be compliant with this object specification. All required attributes must be supported as specified.

**Table 5-39.2 S-Gas Calibration Object Instance Attributes**

Attr ID	Need in Implem	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
1	Optional	Get	NV	Number of Attributes	USINT	Number of attributes supported	
2	Optional	Get	NV	Attribute List	ARRAY OF USINT	List of attributes supported by this object instance	
3	Required	Get	NV	Gas Standard Number	UINT	Gas Type Number	[default] = 0 (no gas type specified) see Semantics section
4	Required	Get	NV	Valid Sensor Instance	UINT	S-Analog Sensor object instance ID for which this object instance is valid	0 = No Valid Sensor n = Instance ID see Semantics section [default] = 0
5	Optional	Set	NV	Gas Symbol	SHORT STRING	Gas Type Name	see Semantics section [default] = null

S-Gas Calibration Object, Class Code: 34<sub>Hex</sub>

Attr ID	Need in Implem	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
6	Optional	Set *	NV	Full Scale	STRUCT of:	Full Scale of the device using this object instance	see Semantics section [default] = 0, 0
					REAL	Amount	The amount of measured parameter corresponding to full scale.
					ENGUNITS	Units	The units for the above. See Data Units Appendix D
7	Optional	Set	NV	Additional Scaler	REAL	Additional Correction Factor	In addition to the correction algorithm, this amount is multiplied to the reading. Generally used for Gas Correction for a gas other than the type identified for the object instance by attribute 3.  (e.g., scale a nitrogen object instance to measure argon).  Default = 1.0
8	Optional	Get	NV	Calibration Date	DATE	Date of Calibration	The date this object instance was last calibrated  [default] = 0
9	Optional	Get	NV	Calibration Gas Number	UINT	Calibration Gas	The gas number of the gas used to calibrate this object instance.  [default] = 0
10	Optional	Get	NV	Gas Correction Factor	REAL	Gas Correction Factor  For devices that support simple correction factors (as opposed to algorithms) for gas selection.	[default] = 1.0
95-96	Defined by Subclasses below						
97-98	Reserved by CIP for Future Use						
99	Conditional **	Get	NV	Subclass	UINT	Identifies a subset of additional instance attributes, services and behaviors.	0 = No subclass 1 = Standard T & P 2 – 65535 = Reserved

\* If this attribute is supported, Set Access is optional.

\*\* If the value of Subclass is 00, then this attribute is OPTIONAL in implementation, otherwise, this attribute is REQUIRED.

### **5-39.2.1 Subclasses**

Each instance level subclass defines a unique meaning for an overlapping range of instance attribute IDs and/or instance service IDs. The range for subclass definitions begins at ID 96 and numbers downward for attributes, and ID 63<sub>hex</sub> and numbers downward for services. The subclass for a given instance is identified by the value of its Subclass instance attribute. The subclasses for this object are specified at the end of this object specification section.

## **5-39.3 Semantics:**

### **5-39.3.1 Gas Standard Number**

Used to identify a gas standard number, for which the object instance is currently calibrated. See Instance Application Example below.

The actual coding of the values are described in the following publication:

See introduction section (above) for reference to the SEMI publication: “Practice for Referencing Gases Used in Digital Mass Flow Controllers”.

Since the actual attributes, and their context, for the parameterization of object instances for particular gas types is beyond the scope of this standard (i.e., vendor specific) the Access Rule for this attribute has been specified as Get. Vendors may choose to specify an Access Rule of Set for this attribute.

### **5-39.3.2 Valid Sensor Instances**

This attribute specifies the S-Analog Sensor object instance for which the S-Gas Calibration object instance is valid. An S-Gas Calibration object instance will be valid for zero or one S-Analog Sensor object instances.

### **5-39.3.3 Gas Symbol**

This optional attribute is a string coded representation of the name of the gas for which the object instance has been configured. It is coded as a user defined text symbol or it is coded as defined in the above referenced SEMI publication.

This attribute may indicate a different gas from the one which has been specified by the *Gas Standard Number*. See Instance Application Example below.

### **5-39.3.4 Full Scale**

This optional attribute identifies the amount of measured parameter (e.g., Mass Flow) corresponding to the Full Scale of the associated S-Analog Sensor object. A primary purpose for this attribute is to allow for simple S-Analog Sensor object implementations where the Value is reported in raw units; this attribute allows a mapping to engineering units.

For example, the Full Scale for a S-Gas Calibration object may be 100 sccm, while the Full Scale for the associated S-Analog Sensor object may be 20,000 counts (i.e., S-Analog Sensor object Data Type = INT and Data Units = Counts).

### 5-39.3.5 Instance Application Example

The following is an example to demonstrate the usage of Gas Calibration object instances and their attributes:

A device has been supplied with three gas calibration object instances: nitrogen (13)2, helium (1) and argon (4). The user wishes to use the device for silane (39) and knows that a correction factor of 0.60 will properly convert a nitrogen calibration for this application. The object instance for nitrogen would be selected and the *Additional Scaler* attribute for this instance would be set to 0.60. To identify this modification, the *Gas Symbol* may be set to read “silane”, “SiH4”, or “39”.

## 5-39.4 Common Services

The S-Gas Calibration Object provides the following Common Services:

**Table 5-39.3 S-Gas Calibration Object Common Services**

Service Code	Need in Implementation		Service Name	Description of Service
	Class	Instance		
0Ehex	Required	Required	Get_Attribute_Single	Returns the contents of the specified attribute.
10hex	Required	Required	Set_Attribute_Single	Modifies an attribute value.

See Appendix A for a description of these services

## 5-39.5 Object-specific Services

**Table 5-39.4 S-Gas Calibration Object-specific Services**

Service Code	Need in Implementation		Service Name	Description of Service
	Class	Instance		
4B <sub>hex</sub>	Required	n/a	Get_All_Instances	Requests a list of all available object instances with their respective gas numbers

**Table 5-39.5 Success Response Service Data Field Parameters**

Parameter	Required	Data Type	Description	Semantics of Values
Size of List	Required	UINT	Specifies the number of elements in the Array	Number of gas calibrations in the list
List of Gas Calibrations	Required if Size > 0	ARRAY of	Supported List	The list of gas calibrations
		STRUCT of	Supported Gas Type	
		UINT	S-Gas Calibration Object Instance ID	
		UINT	Gas Standard Number	

2 The number used with element names shown here is the *Gas Standard Number*

**S-Gas Calibration Object, Class Code: 34<sub>Hex</sub>**

Parameter	Required	Data Type	Description	Semantics of Values
		UINT	Valid Sensor Instance	

**5-39.6 S-Gas Calibration Object Behavior**

The behavior of this object is managed by the Device Supervisor Object, defined in Section 5-35.5.

**5-39.7 S-Gas Calibration Object Instance Subclass 01 (Standard T & P)**

The following specification applies to a subclass of this object for application in Mass Flow Controller devices.

**5-39.7.1 Instance Attributes**

The following Instance Attributes are specified for this object subclass 01.

**Table 5-39.6 Standard T & P Subclass Instance Attributes**

Attr ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
95	Optional	Get	Calibration Pressure	REAL	The gas pressure in KiloPascal Default = 101.32	The Standard Pressure with respect to the calibration conditions.
96	Optional	Get	Calibration Temperature	REAL	The Gas Temperature in Degrees C Default = 0.0	The Standard Temperature with respect to the calibration conditions.

**5-39.8 Services**

There are no additions or restrictions to the Object Services for this object subclass.

**5-39.9 Behavior**

There are no additions or restrictions to the Behavior for this object subclass.

## 5-40 Trip Point Object

**Class Code: 35<sub>Hex</sub>**

The Trip Point Object models the action of trip points for a device, often corresponding to physical outputs (Discrete Output Object). Each Trip Point instance has a source pointer (an attribute of data type Packed EPATH referencing an input value) and a destination pointer (an attribute of data type Packed EPATH referencing a discrete output value). A trip point value, designated as a High or Low trip point, is compared to the specified source value. This trip point is intended to be used as a process control indicator.

### 5-40.1 Class Attributes

**Table 5-40.1 Trip Point Object Class Attributes**

Attribute ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute
1 thru 7	These class attributes are optional and are described in Chapter 4 of this specification.				
97 - 98	Reserved by CIP				
99	Conditional *	Get	Subclass	UINT	Identifies a subset of additional attributes, services and behaviors. The subclasses for an object are specified at the end of the object specification section.

\* If the value of Subclass is 00, then this attribute is OPTIONAL in implementation, otherwise, this attribute is REQUIRED.

#### 5-40.1.1 Subclasses

Each class level subclass defines a unique meaning for an overlapping range of class attribute IDs and/or class service IDs. The range for subclass definitions begins at ID 96 and numbers downward for attributes, and ID 63<sub>hex</sub> and numbers downward for services. The subclass for a given class is identified by the value of its Subclass class attribute. The subclasses for this object are specified at the end of this object specification section.

### 5-40.2 Instance Attributes

**Table 5-40.2 Trip Point Object Instance Attributes**

Attr ID	Need in Implementation	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
1	Optional	Get	NV	Number of Attributes	USINT	Number of attributes supported	
2	Optional	Get	NV	Attribute List	ARRAY OF USINT	List of attributes supported by this object instance	
3	Conditional [At least one of attributes 3 or 5 are required.]	Set	NV	High Trip Point	INT or based Data Type attribute, if supported.	Defines the Value at or above which a trip point condition will occur	[default = 0] See Semantics section

Trip Point Object, Class Code: 35<sub>Hex</sub>

Attr ID	Need in Implementation	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
4	Optional	Set	NV	High Trip Enable	BOOL	Enables the High Trip Point setting.	[default=enabled] See Semantics section
5	Conditional [At least one of attributes 3 or 5 are required.]	Set	NV	Low Trip Point	INT or based Data Type attribute, if supported.	Defines the Value at or below which a trip point condition will occur	[default=0] See Semantics section
6	Optional	Set	NV	Low Trip Enable	BOOL	Enables the Low Trip Point setting.	[default=enabled] See Semantics section
7	Required	Get	V	Status	BOOL	State of this object instance	0 = trip point condition does not exist (unasserted) 1 = trip point condition exists (asserted)  See Semantics section
8	Optional	Set	NV	Polarity	BOOL	Polarity of <i>Output</i> as derived to <i>Status</i> .	0 = Normal ( <i>Output</i> = <i>Status</i> ) 1 = Reverse ( <i>Output</i> = <i>Status</i> inverted)  See Semantics section
9	Optional	Set	NV	Override	USINT	Specifies an override <i>Status</i> .  Note: This attribute may also be set internally by the device during different States.	0 = Normal 1 = Force FALSE or unasserted 2 = Force TRUE or asserted 3 = Freeze <i>Status</i> and <i>Output</i> at existing values 4 – 255 = Reserved by CIP
10	Optional	Set	NV	Hysteresis	Same as High/Low Point Data Type	Determines the amount by which the <i>Input</i> must recover to clear a trip point condition	See Semantics section [default=0]
11	Optional	Set	NV	Delay	UINT	Specifies the amount of time a trip condition must exist before it is reported to <i>Status</i>	Time in milliseconds See Semantics section [default=0]
12	Required	Set	NV	Destination	Packed EPATH	Specifies the path of the destination attribute whose value will be set by <i>Output</i>	See Semantics

Trip Point Object, Class Code: 35<sub>Hex</sub>

Attr ID	Need in Implementation	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
13	Required	Get	V	Output	BOOL	Output of the object the value of which is sent to <i>destination</i>	= <i>Status</i> as a function of <i>Polarity</i> See Semantics
14	Required	Set	NV	Source	Packed EPATH	Specifies the path of the source attribute whose value is retrieved for <i>Input</i>	See Semantics
15	Required	Set	V	Input	INT or specified by Data Type	Input to the object whose value is retrieved from <i>source</i>	See Semantics
16	Optional	Get	NV	Data Units	ENGUNITS	Units of Input, Trip Point, Hysteresis, etc.	See Semantics
17	Optional	Get	NV	Data Type	USINT	Data Type of Input, Trip Point, Hyseresis, etc.	[default] = INT See Semantics
97 - 98	Reserved by CIP						
99	Conditional	Get	NV	Subclass	UINT	Identifies a subset of additional attributes, services and behaviors. The subclasses for this object are specified at the end of this object specification section.	0 = No subclass n = subclass as defined herein

**5-40.2.1 Subclasses**

Each instance level subclass defines a unique meaning for an overlapping range of instance attribute IDs and/or instance service IDs. The range for subclass definitions begins at ID 96 and numbers downward for attributes, and ID 63<sub>hex</sub> and numbers downward for services. The subclass for a given instance is identified by the value of its Subclass instance attribute. The subclasses for this object are specified at the end of this object specification section.

**5-40.3 Semantics****5-40.3.1 High/Low Trip Point, Enable and Status**

*High* or *Low Trip Point* is compared to the *Input* value to generate a trip point condition.

If a single trip point for this instance is required, either the High or Low Trip Point may be enabled, determining not only the direction of the trip point, but also the direction of hysteresis. If two or more separate trip points are required, each exhibiting a separate *Status*, then two or more instances of this object are required.

A trip region can be established with only one instance by enabling both High and Low settings. *Status* will be set if the input value is at or below the *Low Trip Point* OR at or above the *High Trip Point*, cleared if the input value lies between the trip points.

### 5-40.3.2 Hysteresis and Delay

The Delay value specifies how long the trip condition must exist or clear before it is reported in Status. The Hysteresis value specifies the amount by which the Input value must transition in order to clear a trip point condition.

The following relationships demonstrate the logic for a High Trip Point with Hysteresis and assume the use of an internal Timer:

For *Status* not set:

If (*Input* >= *Trip Point High*) and (Timer not running)  
Then start a Timer

If (*Input* >= *Trip Point High*) and (Time >= *Delay*)  
Then set *Status*

If (*Input* < *Trip Point High - Hysteresis*) and (Time < *Delay*)  
Then reset Timer

For *Status* set:

If (*Input* < *Trip Point High - Hysteresis*) and (Timer not running)  
Then start a Timer

If (*Input* < *Trip Point High - Hysteresis*) and (Time >= *Delay*)  
Then clear *Status*

If (*Input* >= *Trip Point High*) and (Time < *Delay*)  
Then clear *Status*

Example: *High Trip Point* = 100, *Hysteresis* = 2, and *Delay* = 1000: will result in a trip point condition being set when the *Input* stays at or above 100 for 1 second and cleared when *Input* drops below 98 for 1 second. Similarly: *Low Trip Point* = 100, *Hysteresis* = 2, and a *Delay* = 1000: will result in a trip point condition being set when *Input* falls at or below 100 for 1 second and cleared when *Input* increases above 102 for 1 second.

### 5-40.3.3 Data Type and Data Units

Specifies the context of *Input* and related attributes (such as *Hysteresis*) for this object instance. Data Units and Data Type will match the source attribute's Data Units and Data Type. See Appendix C for Data Type definitions and Appendix D for Data Units (ENGUNITS) definition.

### 5-40.3.4 Polarity and Output

The value of the *Output* attribute is derived by combining the value of the *Status* attribute with that of the *Polarity* attribute. For a *Polarity* value of Normal, the *Output* will equal the value of *Status*. For a *Polarity* value of Reverse, the *Output* will equal the value of *Status* inverted. The following relationships demonstrate this logic:

For (*Polarity* = 0): *Output* = *Status*  
For (*Polarity* = 1): *Output* = *Status* Inverted

The *Polarity* attribute is optional and is anticipated to be used only to overcome a hardware limitation.

### 5-40.3.5 Source and Input

The *Source* attribute defined as data type Packed EPATH (see CIP Common Specification, Appendix C), specifies the path for the input whose value is retrieved and becomes the value of *Input*.

### 5-40.3.6 Destination and Output

The *Destination* attribute, defined as data type Packed EPATH (see CIP Common Specification, Appendix C), specifies the path for the output whose value is set with the value of *Output*.

## 5-40.4 Common Services

The Trip Point Object provides the following Common Services:

**Table 5-40.3 Trip Point Object Common Services**

Service Code	Need in Implementation		Service Name	Description of Service
	Class	Instance		
0Ehex	Conditional *	Required	Get_Attribute_Single	Returns the contents of the specified attribute.
10hex	n/a	Required	Set_Attribute_Single	Modifies an attribute value.

\* The Get\_Attribute\_Single service is REQUIRED if any attributes are implemented.

See Appendix A for description of these services

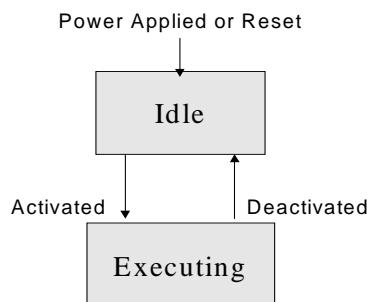
## 5-40.5 Object-Specific Services

The Trip Point Object provides no Object-Specific services.

## 5-40.6 Behavior

### 5-40.6.1 Trip Point Object States

**Figure 5-40.4 Trip Point Object State Transition Diagram**



**Table 5-40.5 DS Instance Behavior State Description**

State	Description
EXECUTING	The Object instance is performing its normal functions.
IDLE	The Object instance is not performing its normal function. The <i>STATUS</i> attribute is NOT ASSERTED

Internal requests are coordinated such that the EXECUTING state here will align with the OPERATIONAL state of the Identity object as well as the EXECUTING state of the S-Device Supervisor object in devices where these objects coexist. Transitions to all other states by these objects will cause this object to transition to its IDLE state.

The Activated and Deactivated events correspond to Identity Object events. Where this object is used with an S-Device Supervisor object (see Chapter 5-35), these events are similarly aligned, but further include alignment with S-Device Supervisor events as specified in Chapter 5-35.

#### 5-40.6.2 Trip Point Object State Event Matrix

**Table 5-40.6 State Event Matrix for S-Device Supervisor Object**

EVENT	STATE	
	Idle	Executing
Power Applied	Default Entry Point	—
Identity Object Transition to Operation state (as by an Activated event)	Transition to EXECUTING State	Ignore Event
Identity Object Transition from Operation state (as by a Deactivated event)	Ignore Event	Transition to IDLE State
S-Device Supervisor Object Transition to Executing State	Transition to EXECUTING State	Ignore Event
S-Device Supervisor Object Transition from Executing State	Ignore Event	Transition to IDLE State

**5-41      Drive Data Object**

**Class Code: Not Assigned**

"This Section is being Reserved for the Drive Data Object."

## 5-42 File Object

**Class Code: 37<sub>hex</sub>**

The number of instances of this object is dependent on the device, and can be obtained by getting the Number\_of\_Instances class attribute. A device may support creation of new instances, and deletion of these dynamically created instances. Instances built into the product shall not be able to be deleted. Instances of the File Object are divided into the following address ranges to provide for publicly defined and product specific files.

**Table 5-42.1 File Object Instance ID Ranges**

Range	Meaning	Quantity
1 – C7 <sub>hex</sub>	Vendor/Product Specific	199
C8 <sub>hex</sub> – FF <sub>hex</sub>	Reserved for CIP (Publicly defined)	56
100 <sub>hex</sub> – 4FF <sub>hex</sub>	Vendor/Product Specific	1024
500 <sub>hex</sub> – FFFF <sub>hex</sub>	Reserved for CIP (Publicly defined)	4,294,966,016

### 5-42.1 Class Attributes

**Table 5-42.2 File Object Class Attributes**

Attr ID	Need in implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1 thru 2	These class attributes are optional and are described in Chapter 4 of this Specification.					
3	Required	Get	Number_of_Instances	UINT	Number of instances present in this device.	0 – 65535
4 thru 7	These class attributes are optional and are described in Chapter 4 of this volume.					
32	Required	Get	Directory	Array of Struct	List of all instance and file names present within this device and the associated instance number.	Number of instance names returned is given in attribute 3, Number_of_Instances. The Instance and File Names are instance attributes 2 and 4, respectively.
			Instance_Number	UINT		
			Instance_Name	STRINGI		
			File_Name	STRINGI		

## 5-42.2 Instance Attributes

**Table 5-42.3 File Object Instance Attributes**

Attr ID	Need in Implementation	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
1	Required	Get	V	State	USINT	State of the Object	0 = Nonexistent 1 = File Empty (no file loaded) 2 = File Loaded 3 = Transfer Upload initiated 4 = Transfer Download initiated 5 = Transfer Upload in Progress 6 = Transfer Download in Progress 7 = Storing 8 – 255 = Reserved
2	Required	Get <sup>2</sup>	NV	Instance Name	STRINGI	Name assigned to this instance.	See Semantics section
3	Required	Get <sup>1</sup>	NV	Instance Format Version	UINT	Format version of the instance.	See Semantics section
4	Required	Set <sup>1</sup>	NV	File Name	STRINGI	Name assigned to file loaded in this instance.	See Semantics section
5	Required	Get <sup>1</sup>	NV	File Revision	USINT USINT	Major_Revision Minor_Revision	See Semantics section
6	Required	Get <sup>1</sup>	NV	File Size	UDINT	Size of the loaded file.	See Semantics section
7	Required	Get <sup>1</sup>	NV	File Checksum	INT	Checksum of the loaded file.	See Semantics section
8	Required	Get <sup>2</sup>	NV	Invocation Method	USINT	Method of invoking downloaded file.	0 = No action required 1 = Reset to Identity Object 2 = Power cycle on device 3 = Start Service request required 4 – 99 = Reserved by CIP 100 – 199 = Vendor Specific 200 – 254 = Reserved by CIP 255 = Not applicable
9	Required	Get	V	File Save Parameters	BYTE	Provides information concerning the nonvolatile storage of the file.	See Semantics section
10	Required	Set	NV	File Type	USINT	Specifies the file type.	0 = Read/Write (default) 1 = Read Only 2 – 255 = Reserved
11	Optional	Get	NV	File Encoding Format	USINT	Defines the encoding format of the data in the file.	See Section 5-42.8.

1. This attribute is updated by way of an object specific service.

2. This attribute is assigned by the product in ‘factory provided’ instances, or assigned by the user during the creation of a new object.

### **5-42.2.1 Semantics**

#### **5-42.2.1.1 Instance\_Name**

This text string describes the meaning of the instance. The name is assigned by either the product (product defined instances) or the user (user created instances). An example of an Instance Name is “Configuration”. All Instance and File names shall be unique across all instances of this class.

#### **5-42.2.1.2 Instance Format Version**

The Instance Format Version value indicates the file format revision of the loaded file. This is a vendor/file specific format, except for the public instances.

#### **5-42.2.1.3 File\_Name**

This text string provided by the user identifies the file loaded for the instance. A File Name example (given an Instance Name = “Configuration”) is “Product mix 4 run”. If a file has not been loaded into the instance, the file name shall be NULL. This attribute is updated upon successful completion of a file download (based on the value provided in the Initiate\_Download service) and may be changed by the user after the download. All Instance and File names shall be unique across all instances of this class.

#### **5-42.2.1.4 File Revision**

This attribute is assigned by the user to identify different revisions of a file for the given instance. If a file is not loaded into the instance, both Major and Minor revision shall be zero. This attribute is updated upon successful completion of a file download based on the value provided in the Initiate\_Download service.

#### **5-42.2.1.5 File\_Size**

This value represents the actual size of the file, in bytes, and is updated at the successful completion of a download based on the total number of bytes transferred. If the transfer fails and the device is not capable of maintaining the previous file (if one existed) the file\_size shall be set to zero.

#### **5-42.2.1.6 File Checksum**

The checksum value is for the entire file. It is the two’s complement of the sixteen-bit sum of all file data.

#### **5-42.2.1.7 File Save Parameters**

This attribute provides information about the nonvolatile save of the file. The bit definition is defined below:

Bit 0:	If set, save is required for nonvolatile storage, otherwise it is saved automatically.
Bits 1 – 3:	Reserved
Bit 4:	Save status; if set loaded file has been saved to nonvolatile storage.
Bits 5 – 7:	Reserved

### 5-42.3 Common Services

**Table 5-42.4 File Object Common Services**

Service Code	Need in Implementation		Service Name	Description of Service
	Class	Instance		
0x06	N/A	Optional	Start	Used to enable file
0x07	N/A	Optional	Stop	Used to disable file
0x08	Optional	N/A	Create <sup>1</sup>	Creates new Upload/Download Object.
0x09	N/A	Optional	Delete <sup>1</sup>	Deletes object previously created using the Create service.
0x0E	Required	Required	Get_Attribute_Single	Returns the contents of the specified attribute.
0x10	N/A	Conditional <sup>2</sup>	Set_Attribute_Single	Writes the contents of the specified attribute.
0x15	N/A	Optional	Restore	Restores a file from nonvolatile storage.
0x16	N/A	Optional	Save	Saves a loaded file to nonvolatile storage.
0x18	N/A	Conditional	Get_Member <sup>3</sup>	Returns a member element of an array attribute

<sup>1</sup>These services shall be supported together.

<sup>2</sup> This service is required when the file type is Read/Write. See error codes in table 5-42.18

<sup>3</sup> This service shall be supported if Identity attribute 12 is implemented.

See Appendix A for complete description of these services

#### 5-42.3.1 Create Service Description

Creates a new file transfer instance in the device. The Instance\_Name parameter is loaded into the Instance\_Name instance attribute of the newly created object. In addition, the State attribute is set to one (Empty, No File Loaded), the File\_Name attribute is set to NULL, and the Revision attribute is set to zero. If a Create request is received with an Instance\_Name, which duplicates an existing Instance or File Name, then an error is returned indicating Invalid Parameter (Error Code 20<sub>hex</sub>).

**Table 5-42.5 Create Request Parameters**

Parameter Name	Data Type	Description of Parameter	Semantics of Values
Instance_Name	STRINGI	Name applied to this instance.	

**Table 5-42.6 Create Response Parameters**

Parameter Name	Data Type	Description of Parameter	Semantics of Values
Instance_Number	UINT	Instance number allocated	0-65535
Invocation_Method	USINT	Method of invoking loaded file.	See Instance Attribute semantics

## 5-42.4 Object-specific Services

**Table 5-42.7 File Object Object-specific Services**

Service Code	Need in Implementation		Service Name	Description of Service
	Class	Instance		
0x4B	N/A	Required	Initiate_Upload	Used to start a file upload
0x4C	N/A	Conditional <sup>1</sup>	Initiate_Download	Used to start a file download
0x4D	N/A	Optional	Initiate_Partial_Read	Used to start a partial read of a file
0x4E	N/A	Optional	Initiate_Partial_Write	Used to start a partial write of a file
0x4F	N/A	Required	Upload_Transfer	Performs a file transfer upload
0x50	N/A	Conditional <sup>1</sup>	Download_Transfer	Performs a file transfer download
0x51	N/A	Conditional <sup>1</sup>	Clear File	Clears a loaded file

<sup>1</sup> These services are required if File Type is Read/Write. Otherwise, they are not supported.

### 5-42.4.1 Initiate\_Upload Service Description

Initiates a file upload. The client indicates the maximum transfer size it can handle. The server indicates total file size and actual transfer size. The device updates the State attribute to ‘Transfer Upload Initiated’ if successful.

**Table 5-42.8 Initiate\_Upload Request Parameters**

Parameter Name	Data Type	Description of Parameter	Semantics of Values
Maximum Transfer Size	USINT	Maximum number of bytes the client can accept in each transfer service.	

**Table 5-42.9 Initiate\_Upload Response Parameters**

Parameter Name	Data Type	Description of Parameter	Semantics of Values
File Size	UDINT	Size of file to upload, in bytes. This value may be the maximum number of bytes, which can be transferred from this instance, and not the actual file size.	
Transfer Size	USINT	Maximum number of bytes the device will deliver in the File_Data parameter of each Upload Transfer response. This value shall be equal to or less than the Maximum Transfer Size specified in the request. The server may send less than this amount.	

### 5-42.4.2 Initiate\_Download Service Description

Initiates a file download. The client indicates total file size, instance format version, file revision, and file name. The server returns the transfer size, number of bytes for each ‘save to non-volatile memory’ and the time required to do the save operation. The device updates the State attribute to ‘Transfer Download Initiated’. If an Initiate\_Download request is received with a File Name, which duplicates an existing Instance or File Name, then an error is returned indicating Invalid Parameter (Error Code 20<sub>hex</sub>).

**Table 5-42.10 Initiate\_Download Request Parameters**

Parameter Name	Data Type	Description of Parameter	Semantics of Values
File Size	UDINT	Size of file to download, in bytes. This value may be the maximum number of bytes, which can be transferred from this instance, and not the actual file size.	
Instance Format Version	UINT	Version of the Instance format, which this file is based on. The server shall verify that this file format is supported.	
File Revision	USINT USINT	Major Revision Minor Revision These values are used to update the File Revision instance attribute after successful completion of the download.	
File Name	STRINGI	User assigned name of file to download. This value is used to update the File_Name instance attribute after successful completion of the download.	

**Table 5-42.11 Initiate\_Download Response Parameters**

Parameter Name	Data Type	Description of Parameter	Semantics of Values
Incremental Burn	UDINT	Number of bytes before a save to non-volatile memory occurs. The save operation may cause the device to delay responding to the transfer which causes a non-volatile save to occur.	
Incremental Burn Time	UINT	Number of seconds required to perform the save to non-volatile memory.	0 – 65535 seconds
Transfer Size	USINT	Maximum number of bytes the device will accept the File_Data parameter of each Download Transfer request. The client may send less than this amount.	

#### 5-42.4.3 **Initiate\_Partial\_Read Service Description**

Initiates a partial file read. The client indicates the file offset, number of bytes to read, and maximum transfer size it can handle. The server shall verify that the entire data range exists in the file. The server returns the read size (which shall match the requested read size) and actual transfer size. The device updates the State attribute to ‘Transfer Upload Initiated’ if successful.

**Table 5-42.12 Initiate\_Partial\_Read Request Parameters**

Parameter Name	Data Type	Description of Parameter	Semantics of Values
File Offset	UDINT	Byte offset into the file where the read begins.	
Read Size	UDINT	Number of bytes to read, starting at File Offset.	
Maximum Transfer Size	USINT	Maximum number of bytes the client can accept in each transfer service.	

The Initiate\_Partial\_Read Response Parameters are identical to the Initiate\_Upload Response parameters.

#### 5-42.4.4 Initiate\_Partial\_Write Service Description

Initiates a partial file write. The client indicates file offset, write size, instance format version, and file revision. The server shall verify that the entire data range exists in the file. The server returns the transfer size, number of bytes for each ‘save to non-volatile memory’, and the time required to do the save operation. The device updates the State attribute to ‘Transfer Download Initiated’.

**Table 5-42.13 Initiate\_Partial\_Write Request Parameters**

Parameter Name	Data Type	Description of Parameter	Semantics of Values
File Offset	UDINT	Byte offset into the file where the write begins.	
Write Size	UDINT	Number of bytes to write, starting at File Offset	
Instance Format Version	UINT	Version of the Instance format, which this file is based on. The server shall verify that this file format is supported.	
File Revision	USINT USINT	Major Revision Minor Revision  These values are used to update the File Revision instance attribute after successful completion of the download.	

The Initiate\_Partial\_Write Response Parameters are identical to the Initiate\_Download Response parameters.

#### 5-42.4.5 Upload\_Transfer Service Description

Performs a file upload. The client and server both verify that an 8-bit value (the Transfer Number), which is sent by the client and returned by the server, is correct (starts at zero and increments by one on each transfer). The State attribute is updated at the first (‘Transfer Upload in Progress’) and last (‘File Loaded’ or ‘Empty, No File Loaded’) transfers. The last transfer packet contains a 16-bit checksum appended to the end of the file data, which is matched by the client to what it calculated. The checksum is not included in the file size.

The server shall send the next packet if the transfer number is one more than the last (or zero and this is either the first packet or a rolled over count), resend the last packet if the transfer number is the same as the last request, or send an error response (error code 20<sub>hex</sub>, Invalid Parameter and Additional Error Code 06<sub>hex</sub>) if the transfer number is neither of the above values.

**Table 5-42.14 Upload\_Transfer Request Parameters**

Parameter Name	Data Type	Description of Parameter	Semantics of Values
Transfer Number	USINT	This 8 bit value starts at zero and is incremented on each additional transfer. The value ‘rolls over’ to zero after 255. The server uses this value to verify the correct packet is received.	0-255=Transfer number

**Table 5-42.15 Upload\_Transfer Response Parameters**

Parameter Name	Data Type	Description of Parameter	Semantics of Values
Transfer Number	USINT	The transfer number is an 8 bit value echoing back the transfer number received.	0 – 255 = Transfer number
Transfer Packet Type	USINT	The transfer packet type provides ‘first’, ‘middle’, and ‘last’ transfer information, as well as the ability to terminate the transfer. The client uses this value to verify integrity of the transfer, and to determine the completion of the transfer.	0 = First transfer packet 1 = Middle transfer packet 2 = Last transfer packet 3 = Abort transfer 4 = First and last packet 5 – 255 = Reserved
File_Data	Array of BYTE	File transfer data.	
Transfer Checksum	INT	Checksum of all transferred file data, only on last transfer.	

**5-42.4.6 Download\_Transfer Service Description**

Performs a file download. The client and server both verify the Transfer Number is correct (starts at zero and increments by one on each transfer). Upon receipt of the first transfer the State attribute is updated to ‘Transfer Download in Progress’. The last transfer packet contains a 16 bit checksum appended to the end of the file data which is matched by the client to what it calculated. The checksum is not included in the file size. Upon completion, the server shall verify the checksum and then update the State attribute (‘File Loaded’ or ‘File Empty’). If the operation was successful, the File Name, File Revision, File Size, and File Checksum attributes are updated. If not, the object transitions to the File Empty state.

When this service is received, the server shall either accept the packet if the transfer number is one more than the last (or zero and this is either the first packet or a rolled over count), discard the packet if the transfer number is the same as the last request, or send an error response (error code 20<sub>hex</sub>, Invalid Parameter and Additional Error Code 06<sub>hex</sub>) if the transfer number is neither of the above values. All error conditions and codes for this service are described in the table below. This object defines one Object Class specific error code.

**Table 5-42.16 Download\_Transfer Request Parameters**

Parameter Name	Data Type	Description of Parameter	Semantics of Values
Transfer Number	USINT	The transfer number is an 8 bit value which starts at zero and is incremented on each additional transfer. The value ‘rolls over’ to zero after 255. The server uses this value to verify integrity of the transfer.	0 – 255 = Transfer number
Transfer Packet Type	USINT	The transfer packet type provides ‘first’, ‘middle’, and ‘last’ transfer information, as well as the ability to terminate the transfer. The server uses this value to verify integrity of the transfer, and to determine the completion of the transfer.	0 = First transfer packet 1 = Middle transfer packet 2 = Last transfer packet 3 = Abort transfer 4 = First and last transfer 5 – 255 = Reserved

**File Object, Class Code: 37<sub>Hex</sub>**

Parameter Name	Data Type	Description of Parameter	Semantics of Values
File_Data	Array of USINT	File transfer data.	
Transfer Checksum	UINT	Checksum of all transferred file data, only on <b>last</b> transfer (Transfer Packet Type = Last).	

**Table 5-42.17 Download\_Transfer Response Parameters**

Parameter Name	Data Type	Description of Parameter	Semantics of Values
Transfer Number	USINT	Transfer number echoed back to client. The client uses this value to verify integrity of the transfer.	0-255=Transfer number

**5-42.4.7 Clear File**

This object-specific instance service clears (deletes contents of) a loaded file from both volatile and nonvolatile memory. There are no parameters defined. In addition to the deletion of the file data, the State is set to File Empty and the following attributes of the instance are cleared:

- File\_Name
- File Revision
- File\_Size
- File\_Checksum
- File\_Save\_Parameters

**5-42.5 Error Codes****5-42.5.1 Error codes for Initiate Services**

The following tables define error codes are used by the Initiate\_Upload, Initiate\_Download, Initiate\_Partial\_Read and Initiate\_Partial\_Write services.

**Table 5-42.18 Error Codes for all Initiate Services**

Error Condition	Error Code	Additional Error Code
OBSOLETE - File Size too large	20 <sub>hex</sub>	00 <sub>hex</sub>
OBSOLETE - Instance Format Version not Compatible	20 <sub>hex</sub>	01 <sub>hex</sub>
File Size too large	20 <sub>hex</sub>	04 <sub>hex</sub>
Instance Format Version not Compatible	20 <sub>hex</sub>	05 <sub>hex</sub>
Fail on transfer – zero size	20 <sub>hex</sub>	08 <sub>hex</sub>
File name too long <sup>1</sup>	15 <sub>hex</sub>	01 <sub>hex</sub>
Too many languages in File Name <sup>1</sup>	15 <sub>hex</sub>	02 <sub>hex</sub>

<sup>1</sup> These errors shall also be used if the condition is detected when the Set\_Attribute\_Single service is used on File Name.

**Table 5-42.19 Error Codes Specific to Initiate Partial Services**

Error Condition	Error Code	Additional Error Code
File Offset out of range	20 <sub>hex</sub>	02 <sub>hex</sub>
Read/Write Size goes beyond end of file	20 <sub>hex</sub>	03 <sub>hex</sub>
File does not exist	02 <sub>hex</sub>	FF <sub>hex</sub>

### 5-42.5.2 Error codes for Transfer Services

The following table defines error codes used by the Download\_Transfer and Upload\_Transfer services.

**Table 5-42.20 Error Codes for Transfer Services**

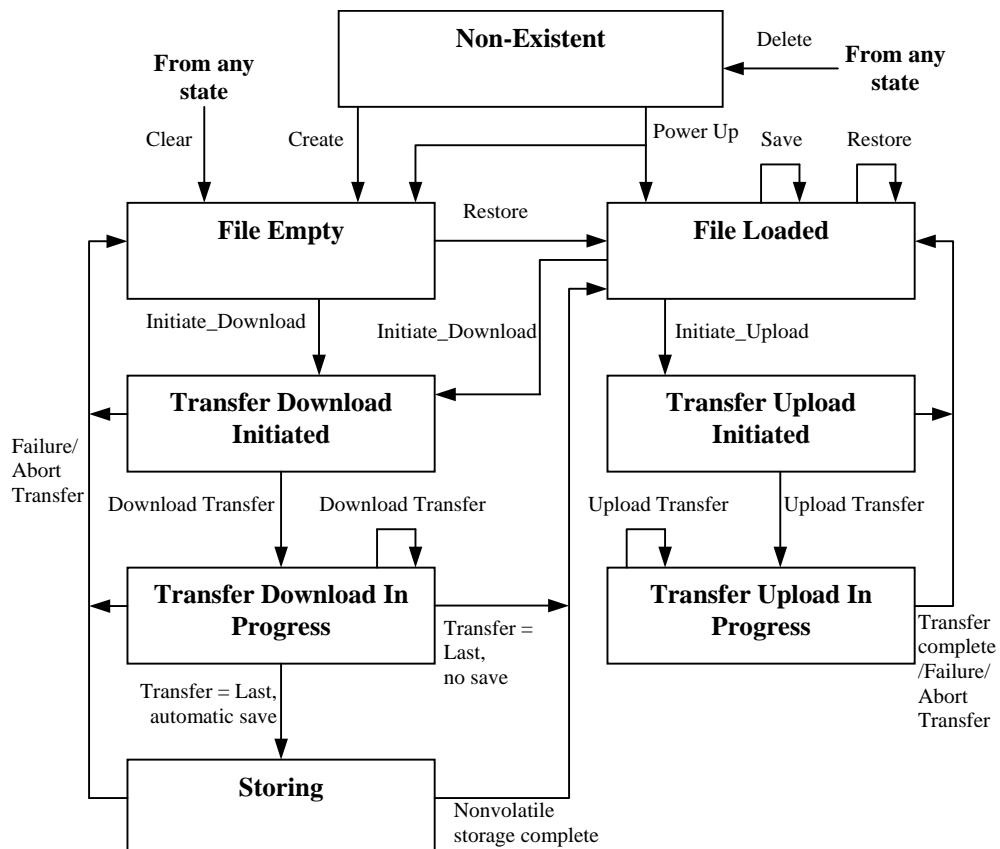
Error Condition	Error Code	Additional Error Code
OBsolete - Fail on transfer – out of sequence	20 <sub>hex</sub>	00 <sub>hex</sub>
OBsolete - Fail on transfer – other	20 <sub>hex</sub>	01 <sub>hex</sub>
Fail on transfer – out of sequence	20 <sub>hex</sub>	06 <sub>hex</sub>
Fail on transfer – other	20 <sub>hex</sub>	07 <sub>hex</sub>
Fail on transfer – zero size	20 <sub>hex</sub>	08 <sub>hex</sub>
Duplicate File Name	20 <sub>hex</sub>	09 <sub>hex</sub>
Fail on checksum	D0 <sub>hex</sub>	FF <sub>hex</sub>
Fail on save to non-volatile memory	19 <sub>hex</sub>	FF <sub>hex</sub>

## 5-42.6 Behavior

### 5-42.6.1 Checksum

The checksum is a way of checking the file transfer's accuracy. It is the two's complement of the sixteen-bit sum of all file data transferred. To quickly determine a checksum value, add up the file transfer data byte's hex values. If the total is greater than 10000<sub>hex</sub>, drop the most significant digit. Then, subtract the result from 10000<sub>hex</sub>. The checksum value provides only a medium level of data security. It cannot detect transposition of bytes during transmission of the file. It also cannot detect the insertion or deletion of the value zero within a packet.

Figure 5-42.21 State Transition Diagram for File Object



## 5-42.6.2 State Event Matrix

Table 5-42.22 File Object State Event Matrix

Event	File Object State							
	Non-Existent	File Empty	Transfer Download Initiated	Transfer Download in Progress	Storing	File Loaded	Transfer Upload Initiated	Transfer Upload in Progress
Power Up	If object exists then If file loaded transition to File Loaded Else transition to File Empty	Not applicable	Not applicable	Not applicable	Not applicable	Not applicable	Not applicable	Not applicable
Create	If Instance Name unique transition to File Empty Else Return error: Invalid Parameter	Not applicable	Not applicable	Not applicable	Not applicable	Not applicable	Not applicable	Not applicable
Initiate Download received <sup>5</sup>	Return error: Object Does Not Exist	Process service data <sup>1</sup> , transition to Transfer Download Initiated	Clear previous service data <sup>2</sup> , process new service data <sup>1</sup>	Clear all file data <sup>3</sup> and attributes <sup>3</sup> , process new service data <sup>1</sup> , transition to Transfer Download Initiated	Return error: Object State Conflict	Clear all file data and attributes <sup>3</sup> , process service data <sup>1</sup> , transition to Transfer Download Initiated	Return error: Object State Conflict	Return error: Object State Conflict
Download Transfer, First Packet <sup>5</sup>	Return error: Object Does Not Exist	Return error: Object State Conflict	Process first block of data, transition to Transfer Download in Progress	Return error: Object State Conflict, clear all file data and attributes <sup>3</sup> , transition to File Empty	Return error: Object State Conflict	Return error: Object State Conflict	Return error: Object State Conflict	Return error: Object State Conflict
Download Transfer, Middle Packet <sup>5</sup>	Return error: Object Does Not Exist	Return error: Object State Conflict	Return error: Object State Conflict, transition to File Empty	Process data	Return error: Object State Conflict	Return error: Object State Conflict	Return error: Object State Conflict	Return error: Object State Conflict
Download Transfer, Last Packet <sup>5</sup>	Not applicable	Return error: Object State Conflict	Return error: Object State Conflict, transition to File Empty	Process data, checksum. If not valid return error: Invalid Parameter and transition to File Empty Else update attribute values and If auto-save transition to Storing Else transition to File Loaded	Return error: Object State Conflict	Return error: Object State Conflict	Return error: Object State Conflict	Return error: Object State Conflict

**File Object, Class Code: 37<sub>Hex</sub>**

<b>Event</b>	<b>File Object State</b>							
	<b>Non-Existent</b>	<b>File Empty</b>	<b>Transfer Download Initiated</b>	<b>Transfer Download in Progress</b>	<b>Storing</b>	<b>File Loaded</b>	<b>Transfer Upload Initiated</b>	<b>Transfer Upload in Progress</b>
<b>Nonvolatile storage complete</b>	Not applicable	Ignore Event	Ignore Event	Ignore Event	Transition to File Loaded	Ignore Event	Ignore Event	Ignore Event
<b>Initiate Upload</b>	Return error: Object Does Not Exist	Return error: Object State Conflict	Return error: Object State Conflict	Return error: Object State Conflict	Return error: Object State Conflict	Process service data <sup>4</sup> , transition to Transfer Upload Initiated	Process new service data <sup>4</sup>	Process new service data <sup>4</sup> , transition to Transfer Upload Initiated
<b>Upload Transfer, First Packet</b>	Return error: Object Does Not Exist	Return error: Object State Conflict	Return error: Object State Conflict	Return error: Object State Conflict	Return error: Object State Conflict	Return error: Object State Conflict	Retrieve first block of data, transition to Transfer Upload in Progress	Return error: Object State Conflict, transition to File Loaded
<b>Upload Transfer, Middle Packet</b>	Return error: Object Does Not Exist	Return error: Object State Conflict	Return error: Object State Conflict	Return error: Object State Conflict	Return error: Object State Conflict	Return error: Object State Conflict	Return error: Object State Conflict, transition to File Loaded	Retrieve data
<b>Upload Transfer, Last Packet</b>	Return error: Object Does Not Exist	Return error: Object State Conflict	Return error: Object State Conflict	Return error: Object State Conflict	Return error: Object State Conflict	Return error: Object State Conflict	Return error: Object State Conflict, transition to File Loaded	Retrieve data and checksum, transition to File Loaded
<b>Clear<sup>5</sup></b>	Return error: Object Does Not Exist	Return success	Clear all file data, Transition to File Empty	Clear all file data, Transition to File Empty	Clear all file data, Transition to File Empty	Clear all file data, Transition to File Empty	Clear all file data, Transition to File Empty	Clear all file data, Transition to File Empty
<b>Transfer Failure/Abort Transfer</b>	Not Applicable	Not applicable	Transition to File Empty	Clear all file data <sup>3</sup> and attributes <sup>3</sup> , transition to File Empty	Not applicable	Not applicable	Transition to File Loaded	Transition to File Loaded
<b>Delete</b>	Return error: Object Does Not Exist	Transition to Non-Existent	Clear all file data <sup>3</sup> and attributes <sup>3</sup> . Transition to Non-Existent	Clear all file data <sup>3</sup> and attributes <sup>3</sup> . Transition to Non-Existent	Clear all file data and attributes <sup>3</sup> . Transition to Non-Existent	Clear all file data and attributes <sup>3</sup> . Transition to Non-Existent	Clear all file data and attributes <sup>3</sup> . Transition to Non-Existent	Clear all file data and attributes <sup>3</sup> . Transition to Non-Existent
<b>Save</b>	Return error: Object Does Not Exist	Return error: Object State Conflict	Return error: Object State Conflict	Return error: Object State Conflict	Return error: Object State Conflict	Save file data and attributes to nonvolatile storage.	Return error: Object State Conflict	Return error: Object State Conflict
<b>Restore</b>	Return error: Object Does Not Exist	Restore file data and attributes from nonvolatile storage, transition to File Loaded.	Return error: Object State Conflict	Return error: Object State Conflict	Return error: Object State Conflict	Clear all file data and attributes <sup>3</sup> . Restore new file data and attributes from nonvolatile storage.	Return error: Object State Conflict	Return error: Object State Conflict

File Object, Class Code: 37<sub>Hex</sub>

Event	File Object State							
	Non-Existent	File Empty	Transfer Download Initiated	Transfer Download in Progress	Storing	File Loaded	Transfer Upload Initiated	Transfer Upload in Progress
<b>Download Transfer, First-and-last Packet</b> <sup>5</sup>	Return error: Object Does Not Exist	Return error: Object State Conflict	Same as Download Transfer First Last Packet in the Download in Progress state	Return error: Invalid Parameter, transition to File Empty	Same as Download Transfer Last Packet	Return error: Object State Conflict	Return error: Object State Conflict	Return error: Object State Conflict
<b>Upload Transfer, First-and-last Packet</b>	Return error: Object Does Not Exist	Return error: Object State Conflict	Return error: Object State Conflict	Return error: Object State Conflict	Return error: Object State Conflict	Return error: Object State Conflict	Same as Upload Transfer Last Packet in the Transfer Upload in Progress state	Return error: Object State Conflict, transition to File Loaded

<sup>1</sup> Processing the Initiate Download service data requires temporary storage of the file name, file size, format version, and file revision and an initialization of the internal transfer variables (file location, transfer number, etc.). If the file name is not unique among all file-instance names associated with the class, then an error is returned indicating Invalid Parameter.

<sup>2</sup> Clearing the previous Initiate Download service data requires deletion of the parameters stored temporarily from the previous Initiate Download service request.

<sup>3</sup> Clearing all file data and attributes requires the deletion of all file data and settable attribute values (file name, file size, etc.) from volatile memory (or from non-volatile memory if data storage is only in non-volatile storage). If the device has only non-volatile storage for this data than the file is lost during this process. If the device has separate volatile and non-volatile storage for the file, it may be recovered by a power cycle or the Restore service.

<sup>4</sup> Processing the Initiate Upload service data requires resolution/retention of the maximum transfer size and an initialization of the internal transfer variables (file location, transfer number, etc.).

<sup>5</sup> When any of these service requests are received and the instance type is Read Only, the error response shall be Object Does not exist for the Non-existent state. In all other states, the error response shall be "Service not supported" if the File type can be modified by the user or "Object state conflict" if the instance is fixed as Read Only.

## 5-42.7 Predefined File Instances

Predefined instances of the File class allow for special purpose files to exist on any product, regardless of vendor or device type. Each predefined file defines non-volatile attribute values, services supported and any other type of behavior. The following table lists these publicly defined files.

**Table 5-42.23 Table of Predefined File Instances**

Instance Number	File Description	Section
C8 <sub>hex</sub>	This Device's EDS File and Icon File	5-42.7.1
C9 <sub>hex</sub>	Collection of related EDS and Icon files	5-42.7.2
CA <sub>hex</sub> – Ff <sub>hex</sub>	Reserved by CIP	

**5-42.7.1 This Device's EDS File and Icon File – (C8<sub>hex</sub>)**

This instance shall contain the EDS file for the device (the device containing this instance), and optionally the Icon file for the device. If this is a device that uses Modular EDSs, then this is the EDS file and Icon for the adapter. One of the following shall be present in this instance:

- Uncompressed EDS file for this device (File Encoding Format 0)
- Compressed EDS file for this device (File Encoding Format 1)
- Compressed EDS and Icon files for this device (File Encoding Format 1)

The following tables describe the services supported and attribute values for this instance.

**Table 5-42.24 Table of Required Attributes for File Instance C8<sub>hex</sub>**

Attribute ID	Access Allowed	Attribute Name	Attribute Value
1	Get	State	See section 5-42.2
2	Get	Instance Name	“EDS and Icon Files”
3	Get	Instance Format Version	1
4	Get	File Name	“EDS.gz” if compressed “EDS.txt” if not compressed
5	Get	File Revision	Shall be the same as the EDS Revision field that is within the EDS file.
6	Get	File Size	See section 5-42.2
7	Get	File Checksum	See section 5-42.2
8	Get	Invocation Method	255
9	Get	File Save Parameters	0
10	Get	File Type	1
11	Get	File Encoding Format	0 or 1

**Table 5-42.25 Table of Service Codes for File Instance C8<sub>hex</sub>**

Service Code	Service Name	Need In Implementation
0x0E	Get_Attribute_Single	Required
0x4B	Initiate Upload	Required
0x4D	Initiate Partial Read	Optional
0x4F	Upload Transfer	Required

### 5-42.7.2 Collection of Related EDS and Icon Files – (C9<sub>hex</sub>)

This instance shall contain a collection of 1 or more EDS and Icon files related to this device, as well as the Icon file for this device if the EDS file (contained in instance C8<sub>hex</sub>) is stored uncompressed. This instance could be applied in support of modular systems, although it is not restricted to that use. This instance shall not contain any files that are included in instance C8<sub>hex</sub> (defined in 5-42.7.1).

The following tables describe the services supported and attribute values for this instance.

**Table 5-42.26 Table of Required Attributes for File Instance C9<sub>hex</sub>**

Attribute ID	Access Allowed	Attribute Name	Attribute Value
2	Get	Instance Name	“Related EDS and Icon Files”
3	Get	Instance Format Version	1
4	Get	File Name	“EDSCollection.gz”
5	Get	File Revision	This is a file revision for the entire collection and not necessarily the revision of any file within the collection.
6	Get	File Size	See section 5-42.2
7	Get	File Checksum	See section 5-42.2
8	Get	Invocation Method	255
9	Get	File Save Parameters	0
10	Get	File Type	1
11	Get	File Encoding Format	1

**Table 5-42.27 Table of Service Codes for File Instance C9<sub>hex</sub>**

Service Code	Service Name	Need In Implementation
0x0E	Get_Attribute_Single	Required
0x4B	Initiate Upload	Required
0x4D	Initiate Partial Read	Optional
0x4F	Upload Transfer	Required

### 5-42.8 File Encoding Format

File data may be encoded (for example, by compressing the data). Attribute 11 (File Encoding Format) of this object shall define the type of encoding. If this optional attribute is not supported there is no encoding on the data or it is unknown.

#### 5-42.8.1 Table of File Encoding Format Values

The defined file encoding formats are listed in the following table.

**Table 5-42.28 Table of File Encoding Format Values**

Attribute Value	Meaning
0	File should be considered as binary – no character, end of line translation or other interpretation should be applied.
1	Compressed file or files. Using the ZLIB compression explained in 5-42.8.2
2 - 255	Reserved

### 5-42.8.2 Compressed File – ZLIB Encoding (01<sub>hex</sub>)

The file shall be encoded with the ZLIB compression file format. ZLIB is a free, general purpose, legally unencumbered, loss-less compression library. The specifications are available from the Internet Engineering Task Force (<http://www.ietf.org>) as:

- RFC 1950, May 1996 (ZLIB Compressed Data Format Specification V.33),
- RFC 1951, May 1996 (DEFLATE Compressed Data Format Specification V. 1.3),
- RFC 1952, May 1996 (GZIP File Format Specification Version 4.3).

A compression and decompression utility and source code are available from the ODVA website ([www.ODVA.org](http://www.ODVA.org)). Additional restrictions on the compressed file(s) are:

- Names of files shall be at least one character long, and no more than 32 characters long.
- Characters shall be ASCII characters: 0-9, a-z, A-Z, “.”, “-”, “\_” case sensitive.
- Duplicate file names are not allowed.
- Directory structure is not supported.
- File attributes and time/date stamps are not supported.

## 5-43 S-partial Pressure Object

**Class Code: 38<sub>Hex</sub>**

An S-Partial Pressure Object provides the means for scanning the partial pressures of a selected Atomic Mass Units (AMU) or gas species. Instances of this object are configured for: AMU, AMU range, gas species or Total Pressure. Class Level services are provided for: requesting a list of configured instances; requesting a list of enabled partial pressures above their respective report thresholds; requesting a list of all enabled partial pressures, irrespective of their report threshold; and, enabling/disabling a group of instances. Also, instance create, delete services are supported.

This object is a member of the Hierarchy of Semiconductor Equipment Devices. As such, its behavior is managed by the S-Device Supervisor Object. See the object definition in Chapter 5.

The S-Partial Pressure object makes use of a list of Standard Gas Type Numbers. This list is described in publication:

SEMI E52-95 “Practice for Referencing Gases Used in Digital Mass Flow Controllers”, Semiconductor Equipment and Materials International (SEMI), Mountain View, CA 94043-4080.

Note: It is implied that the reference above is to the latest revision as specified by SEMI.

### 5-43.1 Class Attributes

**Table 5-43.1 S-partial Pressure Object Class Attributes**

Attribute ID	Need in Implementation	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
1 - 7	These class attributes are optional and are described in Chapter 4 of this specification.						
32	Required	Set	V	Filament Control	BYTE	Enables a filament to turn on if not interlocked	Bit mapped byte see Semantics section
33	Required	Get	V	Filament Status	BYTE	Indicates that a filament is on or off and if a filament fault condition exists	Bit mapped byte see Semantics section
34	Required	Get	V	Reading Invalid	BOOL	A device will indicate Reading Invalid if it determines that one or more of its operational parameters are beyond factory set limits.	0 = Valid 1 = Invalid
35	Optional	Set	NV	Emission Current	REAL	Filament emission current	AMPS

**S-Partial Pressure Object, Class Code: 38<sub>Hex</sub>**

Attribute ID	Need in Implementation	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
36	Optional	Set	V	Multiplier On	BOOL	Turns on the electron multiplier	0 = Off 1 = On
37	Optional	Set	NV	Multiplier Voltage	REAL	Sets the multiplier field excitation voltage	VOLTS
38	Optional	Set	NV	Samples	USINT	specifies the number of samples configured for each enabled instance	0 = manufacturer default [default]
39	Optional	Set	NV	Scan Count	UINT	increments by one upon completion of each Measurement Scan **	0 - 65535 (value after 65535 is 0)
97 - 98	Reserved by CIP for Future Use						
99	Conditional *	Get	NV	Subclass	UINT	Identifies a subset of additional attributes, services and behaviors. The subclasses for this object are specified at the end of this object specification section	0 = No Subclass 1 – 65535 = Reserved

\* If the value of this attribute is 0, then its support is OPTIONAL, otherwise support is REQUIRED

\*\* A Measurement Scan is considered to be one complete cycle of measurements performed by the Device.

### 5-43.1.1 Class Level Subclasses

Each class level subclass defines a unique meaning for an overlapping range of class attribute IDs and/or class service IDs. The range for subclass definitions begins at ID 96 and numbers downward for attributes, and ID 63<sub>hex</sub> and numbers downward for services. The subclass for a given class is identified by the value of its Subclass class attribute. The subclasses for this object are specified at the end of this object specification section.

### 5-43.1.2 Semantics

#### 5-43.1.2.1 Filament Control

A bit that is set to 1 (one) in the *Control* byte enables the corresponding filament to turn on. Once enables, a filament will turn on if it is able. Various external or internal interlocks may prevent the filament from turning on.

**Table 5-43.2 Filament Control Attribute**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	Reserved	Reserved	Reserved	Filament 4 Enable	Filament 3 Enable	Filament 2 Enable	Filament 1 Enable

**S-Partial Pressure Object, Class Code: 38<sub>Hex</sub>**

An attempt to set a *Filament Enable* bit corresponding to a filament not supported by the device will return an *Invalid Attribute Value* error, as will an attempt to set a filament state not supported by the device (e.g., multiple filaments enabled).

**5-43.1.2.2 Filament Status**

If a filament is on, the corresponding *Filament On* bit in the *Status* byte is set to 1 (one). If a filament has been determined by the device to be in a fault state (such as a burned out or broken filament), the corresponding *Filament Fault* bit on the *Status* byte is set to 1 (one).

**Table 5-43.3 Filament Status Attribute**

<b>Bit 7</b>	<b>Bit 6</b>	<b>Bit 5</b>	<b>Bit 4</b>	<b>Bit 3</b>	<b>Bit 2</b>	<b>Bit 1</b>	<b>Bit 0</b>
Filament 4 Fault	Filament 3 Fault	Filament 2 Fault	Filament 1 Fault	Filament 4 On	Filament 3 On	Filament 2 On	Filament 1 On

**5-43.2 Instance Attributes**

Certain minimal implementations may support some optional “Set” attributes as “Get” only and still be compliant with this object specification. All required attributes must be supported as specified.

**Table 5-43.4 S-partial Pressure Object Instance Attributes**

<b>Attr ID</b>	<b>Need in Implementation</b>	<b>Access Rule</b>	<b>NV</b>	<b>Name</b>	<b>Data Type</b>	<b>Description of Attribute</b>	<b>Semantics of Values</b>
1	Optional	Get	V	Number of Attributes	USINT	Number of supported attributes	The number of attributes supported by this object instance
2	Optional	Get	V	Attribute List	ARRAY OF USINT	List of supported attributes	The list of attributes supported by this object instance
3	Required	Set	NV	Instance Type	USINT	Specifies the measurement type for the instance	0 = AMU [default] 1 = AMU Range 2 = Gas Species 3-50 = reserved 51-99= Device Specific 100-255 = Vendor Specific
4	Conditional *	Set	NV	AMU	REAL	Atomic Mass Unit (or Starting AMU for analog)	[default] = 0 see Semantics section
5	Conditional *	Set	NV	Ending AMU	REAL	Ending AMU for an AMU Range type instance	see Semantics section
6	Conditional *	Set	NV	Gas Standard Number	UINT	Gas Species Type Number	[default] = 0 (no gas type specified) see Semantics section
7	Required	Get	V	Status	BYTE	Alarm and Warning State of this object instance	see Semantics section

S-Partial Pressure Object, Class Code: 38<sub>Hex</sub>

Attr ID	Need in Implementation	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
8	Optional	Set	NV	Alarm High Enable	BOOL	Enables the setting of the Alarm High Status Bits	0 = disable [default] 1 = enable
9	Optional	Set	NV	Alarm Low Enable	BOOL	Enables the setting of the Alarm Low Status Bits	0 = disable [default] 1 = enable
10	Optional	Set	NV	Warning High Enable	BOOL	Enables the setting of the Warning High Status Bits	0 = disable [default] 1 = enable
11	Optional	Set	NV	Warning Low Enable	BOOL	Enables the setting of the Warning Low Status Bits	0 = disable [default] 1 = enable
12	Required	Set	NV	Instance Enable	USINT	Enables the measurement of this AMU	0 = disable [default] 1 = normal scan rate 2 = double scan rate 3-50 = reserved 51-99= Device Specific 100-255 = Vendor Specific  see Semantics section
13	Required	Get	V	Partial Pressure	REAL	The measured Partial Pressure of the specified AMU	The measured value of partial pressure for this AMU.  Units are specified by attribute <i>Pressure Units</i> .
14	Optional	Set	NV	Pressure Units	ENGUNITS	Determines the <i>units</i> context of the <i>Partial Pressure</i> attribute	see Appendix D for a list of values.  [default] = Torr  If attribute is not supported, [default] is implied.
15	Optional	Set	NV	Dwell Time	UINT	Determines the acquisition time for this AMU	Time in Milliseconds 0 = fastest possible [default]
16	Optional	Set	NV	Electron Energy	REAL	Indicates the Electron Impact Ionization Voltage	Units = eV  [default] = Factory Configured

S-Partial Pressure Object, Class Code: 38<sub>Hex</sub>

Attr ID	Need in Implementation	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
17	Optional	Set	NV	Report Threshold	REAL	Trip point for reporting in the Get_Pressures service response	The level above which the <i>Partial Pressure</i> will be reported. see Class Level Services [default] = 0
18	Optional	Set	NV	Alarm Trip Point High	REAL	Determines the Value above which an Alarm Condition will occur	see Semantics section [default] = Maximum value for its data type.
19	Optional	Set	NV	Alarm Trip Point Low	REAL	Determines the Value below which an Alarm Condition will occur	see Semantics section [default] = Minimum value for its data type.
20	Optional	Set	NV	Alarm Hysteresis	REAL	Determines the amount by which the <i>Value</i> must recover to clear an Alarm Condition	see Semantics section [default] = 0
21	Optional	Set	NV	Warning Trip Point High	REAL	Determines the Value above which a Warning Condition will occur	see Semantics section [default] = Maximum value for its data type.
22	Optional	Set	NV	Warning Trip Point Low	REAL	Determines the Value below which a Warning Condition will occur	see Semantics section [default] = Minimum value for its data type.
23	Optional	Set	NV	Warning Hysteresis	REAL	Determines the amount by which the <i>Value</i> must recover to clear a Warning Condition	see Semantics section [default] = 0
24	Optional	Set	NV	Group ID	USINT	Identifies this instance as belonging to the specified group	0 - 255 groups
97 - 98	Reserved by CIP for Future Use						

**S-Partial Pressure Object, Class Code: 38<sub>Hex</sub>**

Attr ID	Need in Implementation	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
99	Conditional **	Get	NV	Subclass	UINT	Identifies a subset of additional attributes, services and behaviors. The subclasses for this object are specified at the end of this object specification section.	0 = No Subclass 1 - 65535 = Reserved

\* Which of these parameterization attributes are supported depends upon the allowable values for the *Instance Type* attribute.

\*\* If the value of Subclass is 00, which identifies "no subclass", then this attribute is OPTIONAL in implementation, otherwise, this attribute is REQUIRED.

### 5-43.2.1 Instance Level Subclasses

Each instance level subclass defines a unique meaning for an overlapping range of instance attribute IDs and/or instance service IDs. The range for subclass definitions begins at ID 96 and numbers downward for attributes, and ID 63<sub>hex</sub> and numbers downward for services. The subclass for a given instance is identified by the value of its Subclass instance attribute. The subclasses for this object are specified at the end of this object specification section.

### 5-43.2.2 Semantics

#### 5-43.2.2.1 Instance Type

The vendor must specify which values are supported. An attempt to set this attribute to an unsupported value will return an Invalid Attribute Value error.

#### 5-43.2.2.2 AMU

The *AMU* attribute is used to configure the instance for a particular Atomic Mass Unit. For basic AMU type configurations, this specifies the point at which the partial pressure is determined. For AMU Range type configurations, this specifies the point at which the range of AMU begins. A value of zero (0) configures the object instance for Total Pressure. Total Pressure is defined as the sum of all enabled S-Partial Pressure object instances. For Gas Species type configurations, this value is ignored.

#### 5-43.2.2.3 Ending AMU

For basic AMU type configurations, this value is ignored. For AMU Range type configurations, this specifies the point at which the range of AMU ends. For Gas Species type configurations, this value is ignored.

#### 5-43.2.2.4 Gas Standard Number

The Gas Standard Number attribute is used to configure the instance for a particular gas species. For basic AMU, and AMU Range, type configurations, this value is ignored. For Gas Species type configurations, this specifies the Gas Standard Number for the particular gas whose partial pressure is being measured and reported.

The actual coding of the values are described in the following publication:

See introduction section (above) for reference to the SEMI publication:  
“Practice for Referencing Gases Used in Digital Mass Flow Controllers”.

#### 5-43.2.2.5 Status

Status is a bit mapped byte, which indicates the Alarm and Warning Exception status of the object instance. The following definition applies:

**Table 5-43.5 Status Attribute**

Bit	Definition
0	Alarm High Exception: 0 = cleared; 1 = set
1	Alarm Low Exception: 0 = cleared; 1 = set
2	Warning High Exception: 0 = cleared; 1 = set
3	Warning Low Exception: 0 = cleared; 1 = set
4	Reserved
5	Reserved
6	Reserved
7	Reserved

#### 5-43.2.2.6 Instance Enable

The *Instance Enable* attribute is used to turn on/off the object instance. Although an instance may be completely configured, if *disabled*, the device will skip the partial pressure measurement in its scan routine.

Due to the time dependent nature of measuring a partial pressure, a feature is included that allows for a double scan rate. This mode of operation causes the device to measure this AMU twice as often as it normally would. This yields a faster update rate for this AMU’s partial pressure measurement.

#### 5-43.2.2.7 Dwell Time and Electron Energy

In the event that one of these values is modified while the device is actively acquiring a corresponding reading, the measurement in process shall proceed as originally configured and the new values shall take effect on subsequent readings.

#### 5-43.2.2.8 Trip Points and Hysteresis

A Trip Point High value is that value above which the *Value* attribute will cause an Alarm or Warning exception condition.

A Trip Point Low value is that value below which the *Value* attribute will cause an Alarm or Warning exception condition.

**S-Partial Pressure Object, Class Code: 38<sub>Hex</sub>**

A Hysteresis value specifies the amount by which the *Value* attribute must transition in order to clear an Alarm or Warning condition. For example: A Trip Point High value of 100 and a hysteresis value of 2 will result in an exception condition being set when the *Partial Pressure* is above 100 and cleared when the *Partial Pressure* drops below 98. Similarly, A Trip Point Low value of 100 and a hysteresis value of 2 will result in an exception condition being set when the *Partial Pressure* is below 100 and cleared when the *Partial Pressure* increases above 102.

### 5-43.3 Common Services

**Table 5-43.6 S-partial Pressure Object Common Services**

Service Code	Need in Implementation		Service Name	Description of Service
	Class	Instance		
01 <sub>hex</sub>	n/a	Optional	Get_Attributes_All	Returns a predefined listing of this objects attributes (See the Get_Attributes_All Response definition below)
08 <sub>hex</sub>	Optional	n/a	Create	Requests an instantiation of a new object within this class
09 <sub>hex</sub>	Optional *	Optional	Delete	Deletes an object instance
0E <sub>hex</sub>	Conditional **	Required	Get_Attribute_Single	Returns the contents of the specified attribute.
10 <sub>hex</sub>	n/a	Required	Set_Attribute_Single	Modifies an attribute value.

\* The Delete service request at the class level deletes all instances.

\*\*The Get\_Attribute\_Single service is REQUIRED if any attributes are implemented.

See Appendix A for definitions of these common services.

#### 5-43.3.1 Get\_Attributes\_All Response

At the **Class level** this service is not supported.

At the **Instance level**, the order of the attributes returned in the “Object/Service specific reply data” portion of the Get\_Attributes\_All response is as follows:

**Table 5-43.7 Get\_Attributes\_All Response Service Data**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Number of Attributes Default = 0 [specifies N bytes]							
1	Attribute List (attribute # 1)							
N	Attribute List (attribute # N)							
N+1 thru End	Attributes 3-99 as specified in the above Attribute List							

**Important:** If the instance attribute “Number of Attributes” is not supported, the default value of zero is to be inserted in its place and **no** members of the “Attribute List” attribute will follow.

**Important:** Default values for all unsupported attributes are inserted.

## 5-43.4 Object-specific Services

**Table 5-43.8 S-partial Pressure Object-specific Services**

Service Code	Need in Implementation		Service Name	Description of Service
	Class	Instance		
4B <sub>hex</sub>	Required	n/a	Create_Range	Requests a Create for range of instances. See description below.
4C <sub>hex</sub>	Required	n/a	Get_Instance_List	Requests a list of all enabled object instances with their respective AMU numbers
4D <sub>hex</sub>	Optional	n/a	Get_Pressures	Requests a list of all enabled Partial Pressures above their respective thresholds
4E <sub>hex</sub>	Optional	n/a	Get_All_Pressures	Requests a list of all enabled Partial Pressures
4F <sub>hex</sub>	Optional	n/a	Group_Enable	Enables or Disables a group of instances as identified by their respective <i>Group ID</i> attribute values

### 5-43.4.1 Create\_Range Service

**Table 5-43.9 Create\_Range Request Service Data Field Parameters**

Parameter	Required	Data Type	Description
Start AMU	Required	REAL	Specifies the AMU value to be assigned to the first instance
End AMU	Required	REAL	Specifies the AMU value to be assigned to the last instance
Number of Instances	Required	UINT	Specifies the number of instances to be created
Starting ID	Required	UINT	Specifies the first instance ID in the list to be created. 0 = use next available
Group ID	Optional	USINT	Specifies the Group value to assign to the created instances.

The successful response to a Create\_Range service request is the creation of multiple S-Partial Pressure object instances. The device automatically sets the AMU value of each instance such that the range of AMU specified in the request is covered in evenly spaced increments by the specified number of instances.

If the Starting ID parameter value is zero (0), the device will start with the first available instance ID that allows for the Number of Instances specified to be created in one contiguous ID numbered range.

**Table 5-43.10 Create\_Range Success Response Service Data Field Parameters**

Parameter	Data Type	Description
Starting ID	UINT	Indicates the first instance ID in the list created
Ending ID	UINT	Indicates the last instance ID in the list created

#### 5-43.4.2 Get\_Instance\_List Service

The Get\_Instance\_List service request has no service data field parameters.

**Table 5-43.11 Get\_Instance\_List Success Response Service Data Field Parameters**

Parameter	Required	Data Type	Description	Semantics of Values
Size of List	Required	UINT	Specifies the number of elements in the Array	Number of object instances in the list
List of Gas Calibrations	Required if Size > 0	ARRAY of	Supported List	The list of enabled object instances
		STRUCT of		
		UINT	Instance Number	
		USINT	Instance Type	
		REAL	AMU	
		REAL	Ending AMU	
		UINT	Gas Standard Number	

#### 5-43.4.3 Get\_Pressures Service

The Get\_Pressures service request has no service data field parameters.

**Table 5-43.12 Get\_Pressures Success Response Service Data Field Parameters**

Parameter	Required	Data Type	Description	Semantics of Values
Size of List	Required	UINT	Specifies the number of elements in the Array	Number of Partial Pressures in the list
List of Gas Calibrations	Required if Size > 0	ARRAY of	Supported List	The list of all enabled Partial Pressures above their respective thresholds
		STRUCT of		
		UINT	Instance ID	
		REAL	Partial Pressure	

#### 5-43.4.4 Get\_All\_Pressures Service

The Get\_All\_Pressures service request has no service data field parameters.

**Table 5-43.13 Get\_All\_Pressures Success Response Service Data Field Parameters**

Parameter	Required	Data Type	Description	Semantics of Values
Size of List	Required	UINT	Specifies the number of elements in the Array	Number of Partial Pressures in the list
List of Gas Calibrations	Required if Size > 0	ARRAY of	Supported List	The list of all enabled Partial Pressures
		STRUCT of		
		UINT	Instance ID	
		REAL	Partial Pressure	

**5-43.4.5 Group\_Enable Service****Table 5-43.14 Group\_Enable Request Service Data Field Parameters**

Parameter	Required	Data Type	Description
Enable	Required	BOOL	0 = Disable 1 = Enable
Group ID	Required	USINT	Specifies the Group (i.e., all S-Partial Pressure object instances whose Group ID attribute has this value is effected by this service request)  An unrecognized Group parameter returns an Invalid Parameter error.

The Group\_Enable Service Reply has no service data field parameters.

**5-43.5 Object Behavior**

The behavior of this object is managed by the S-Device Supervisor Object, defined in Section 5-35.

## 5-44 S-Sensor Calibration Object

### Class Code: 40<sub>Hex</sub>

An S-Sensor Calibration Object affects the behavior of an associated S-Analog Sensor object instance; a device profile will show a relationship between these two objects where they are used. The S-Analog Sensor object uses the Calibration Object Instance attribute (Attribute ID 35) as a selection mechanism. The S-Sensor Calibration Object provides the data with which a device enacts the appropriate calibration algorithm for a given application. Usually, these calibration objects are instantiated in correspondence to various application parameters for a particular sensor type, for example fluid type. Each S-Sensor Calibration Object Instance contains a set of attribute values for one particular calibration set.

The S-Sensor Calibration class level object provides a service for retrieving a list of all valid object instances. The service response includes a list of elements. Each element includes: Instance ID, Calibration ID Number and the valid S-Analog Sensor object instance ID for which the instance is valid.

There may be more than one instance with the same Calibration ID Number. These instances may be differentiated by Full Scale, Calibration Name, Additional Scaler and/or other parametric distinctions, including valid S-Analog Sensor object instance ID. The distinctions may, or may not, be evident in the Get\_All\_Instances service response, depending upon what the distinction is.

S-Sensor Calibration Objects most often utilize the region of Manufacturer Defined Coefficients Attributes (ID 16-31) for specific calibration parameters. If more coefficients are needed, then they may be placed into the area of ID >100 – the vendor-specific range.

This object is a member of the *Hierarchy of Semiconductor Equipment Devices*. As such, its behavior is managed by the Device Supervisor Object. See Section 5-35 of this volume for more information.

### 5-44.1 Class Attributes

**Table 5-44.1 S-Sensor Calibration Object Class Attributes**

Attribute ID	Need in implementation	Access Rule	Name	Data Type	Description of Attribute
1 thru 7	These class attributes are either optional or conditional and are described in Chapter 5 of this specification.				
97-98	Reserved by CIP for Future Use				
99	Conditional *	Get	Subclass	UINT	Identifies a subset of additional class attributes, services and behaviors.

\* If the value of Subclass is 00, which identifies "no subclass", then this attribute is OPTIONAL in implementation, otherwise, this attribute is REQUIRED.

## 5-44.2 Subclasses

Each class level subclass defines a unique meaning for an overlapping range of class attribute IDs and/or class service IDs. The range for subclass definitions begins at ID 96 and numbers downward for attributes, and ID 63<sub>hex</sub> and numbers downward for services. The subclass for a given class is identified by the value of its Subclass class attribute. The subclasses for this object are specified at the end of this object specification section.

## 5-44.3 Instance Attributes

Certain minimal implementations may support any optional “Set” attributes as “Get” only and still be compliant with this object specification. All required attributes must be supported as specified.

**Table 5-44.2 S-Sensor Calibration Object Instance Attributes**

Attr ID	Need in Implementation	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
1	Optional	Get	NV	Number of Attributes	USINT	Number of attributes supported	
2	Optional	Get	NV	Attribute List	ARRAY OF USINT	List of attributes supported by this object instance	
3	Required	Get	NV	Calibration ID Number	UINT	Identifies the Calibration (e.g., fluid type)	see Semantics section [default] = 0
4	Required	Get	NV	Valid Sensor Instance	UINT	S-Analog Sensor object instance ID for which this object instance is valid	0 = No Valid Sensor n = Instance ID see Semantics section [default] = 0
5	Optional	Set	NV	Calibration Name	SHORT STRING	ASCII Text, max 50 characters, representation of the calibration (e.g. material type)	see Semantics section [default] = null
6	Optional	Get	NV	Full Scale	STRUCT of:	Full Scale of the device using this object instance	see Semantics section [default] = 0, 0
					REAL	Amount	The amount of measured parameter corresponding to full scale.
					ENGUNITS	Units	The units for the above. see Data Units Appendix K

S-Sensor Calibration Object, Class Code: 40<sub>Hex</sub>

Attr ID	Need in Implementation	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
7	Optional	Set	NV	Additional Scaler	REAL	Additional Correction Factor	In addition to the correction algorithm, this amount is multiplied to the reading. Default = 1.0
8	Optional	Get	NV	Calibration Date	DATE	Date of Calibration	The date this object instance was last calibrated [default] = 0
9	Optional	Set	NV	Temperature Data Units	ENGUNITS	Determines the Units context of the Temperature attribute	see Semantics section
10	Optional	Set	NV	Temperature	REAL	Temperature	see Semantics section
11	Optional	Set	NV	Pressure Data Units	ENGUNITS	Determines the Units context of the Pressure attribute	see Semantics section
12	Optional	Set	NV	Pressure	REAL	Pressure	see Semantics section
13	Optional	Set	NV	Coefficient A	REAL	The "a" term, in a quadratic correction	$y = ax^2 + bx + c$
14	Optional	Set	NV	Coefficient B	REAL	The "b" term, in a quadratic correction	$y = ax^2 + bx + c$
15	Optional	Set	NV	Coefficient C	REAL	The "c" term, in a quadratic correction	$y = ax^2 + bx + c$
16-31	Optional	Set	NV	Manufacturer Defined Coefficients **	**	**	**
95-96	Defined by Subclasses below						
97-98	Reserved by CIP for Future Use						
99	Conditional *	Get	NV	Subclass	UINT	Identifies a subset of additional instance attributes, services and behaviors.	0 = No subclass 1 – 65535 = Reserved

\* If the value of Subclass is 00, then this attribute is OPTIONAL in implementation, otherwise, this attribute is REQUIRED.

\*\* These attributes (16-31) are set aside for manufacturer specified calibration coefficients. This is to encourage the limitation and placement of these to facilitate the user's ability to download, upload and otherwise verify these values.

#### **5-44.4 Subclasses**

Each instance level subclass defines a unique meaning for an overlapping range of instance attribute IDs and/or instance service IDs. The range for subclass definitions begins at ID 96 and numbers downward for attributes, and ID 63<sub>hex</sub> and numbers downward for services. The subclass for a given instance is identified by the value of its Subclass instance attribute. The subclasses for this object are specified at the end of this object specification section.

#### **5-44.5 Semantics**

##### **5-44.5.1 Calibration ID Number**

Used to identify a particular calibration using a standardized identifying number, for which the object instance is currently calibrated. See Instance Application Example below.

The actual coding of these values are specified in the Device Profile or vendor specific implementation where used. Since the actual attributes, and their context, for the parameterization of object instances for particular calibration types is beyond the scope of this standard (i.e., vendor specific) the Access Rule for this attribute has been specified as Get. Vendors may choose to specify an Access Rule of Set for this attribute.

##### **5-44.5.2 Valid Sensor Instances**

This attribute specifies the S-Analog Sensor object instance for which the S-Sensor Calibration object instance is valid. An S-Sensor Calibration object instance will be valid for zero or one S-Analog Sensor object instances.

##### **5-44.5.3 Calibration Name**

This optional attribute is a string coded representation of the name of the calibration for which the object instance has been configured. It is coded as a user defined text symbol or it is coded as defined in the Device Profile where used.

##### **5-44.5.4 Full Scale**

This optional attribute identifies the amount of measured parameter (e.g., Flow Rate) corresponding to the Full Scale of the associated S-Analog Sensor object. A primary purpose for this attribute is to allow for simple S-Analog Sensor object implementations where the Value is reported in raw units; this attribute allows a mapping to engineering units.

For example, the Full Scale for a S-Sensor Calibration object may be 100 LPM, while the Full Scale for the associated S-Analog Sensor object may be 20,000 counts (i.e., S-Analog Sensor object Data Type = INT and Data Units = Counts).

### 5-44.5.5 Temperature and Pressure

These optional attributes are used for Calibrations where a temperature and/or a pressure dependency is known. These attributes may be supported with "set access" to allow the user to configure the calibration object or to continuously feed the value from the network. Alternatively, these attributes may be supported with "get access" to identify the temperature and/or pressure for which the calibration has been set. Another alternative is that these attributes may be shown with a linkage in a Device Profile whereby the value is modified internal to the device and would again be shown with "get access" to the network. In the latter case, the related Data Units attributes may not be supported.

### 5-44.5.6 Instance Application Example

The following is an example to demonstrate the usage of S-Sensor Calibration object instances and their attributes:

A device has been supplied with three S-Sensor Calibration object instances: DI Water, Ethylene Glycol 50%, and Ethylene Glycol 70%. The user wishes to use the device for ethylene glycol 80% and knows that a correction factor of 1.05 will properly convert the ethylene glycol 70% calibration for this application. The object instance for ethylene glycol 70% would be selected and the *Additional Scaler* attribute for this instance would be set to 1.05. To identify this modification, the *Calibration Name* may be set to read “Ethylene Glycol 80%”, “HOCH<sub>2</sub>CH<sub>2</sub>OH”, or perhaps, “Coolant X”.

## 5-44.6 Common Services

The S-Sensor Calibration Object provides the following Common Services:

**Table 5-44.3 S-Sensor Calibration Object Common Services**

Service Code	Need in Implementation		Service Name	Description of Service
	Class	Instance		
0E <sub>hex</sub>	Conditional*	Required	Get_Attribute_Single	Returns the contents of the specified attribute.
10 <sub>hex</sub>	Conditional**	Required	Set_Attribute_Single	Modifies an attribute value.

\* Required if any attributes are supported.

\*\* Required if any settable attributes are supported.

See Appendix A for definition of these services

## 5-44.7 Object-specific Services

**Table 5-44.4 S-Sensor Calibration Object-specific Services**

Service Code	Need in Implementation		Service Name	Description of Service
	Class	Instance		
4B <sub>hex</sub>	Required	n/a	Get_All_Instances	Requests a list of all available object instances with their respective Calibration ID Numbers

**5-44.7.1 Get\_All\_Instances Request Service Data Field Parameters**

None.

**5-44.7.2 Get\_All\_Instances Success Response****Table 5-44.5 S-Get\_All\_Instances Success Response Service Data Field Parameters**

Parameter	Required	Data Type	Description	Semantics of Values
Size of List	Required	UINT	Specifies the number of elements in the Array	Number of calibrations in the list
List of Calibrations	Required if Size > 0	ARRAY of	Supported List	The list of Sensor Calibrations
		STRUCT of	Supported Calibration	
		UINT	S-Sensor Calibration Object Instance ID	
		UINT	Calibration ID Number	
		UINT	Valid Sensor Instance	

**5-44.8 S-Sensor Calibration Object Behavior**

The behavior of this object is managed by the S-Device Supervisor Object, defined in Section 5-35.5.

## 5-45 Event Log Object

**Class Code: 41<sub>Hex</sub>**

The Event Log Object provides event indication and/or event logging on CIP nodes. Some applications may only require event indication, while others may only require event logging.

Event information is stored in arrays and these arrays have an assumed index equal to the element number in the array.

Multiple object instances are only required for devices which support independent logs. Each Event Log Object instance has an event instance identifier (user-defined number and optional name).

Each Event Log Object instance maintains a unique event log, which may contain time-stamped application data for that set of events. Each event associated with an instance may be assigned a unique identifier. An attribute of the Event Log Object Instance defines all of the events associated with the particular instance. Another attribute indicates the current state of each event. An event log entry occurs upon a state change in the status of any single event in the set of events being monitored by the particular Event Log Object Instance. Data stored within the event log is configurable to include application-specific data and time / date, in addition to the event identifier. The application data is optionally sent along with the event trigger from the application object to the Event Log Object. Both the event log size and its storage trigger mechanism are configurable.

The Event Log Object class maintains a list of the Event Log Object Instances.

The cause of an event is not defined for the Event Log Object. A local application is the event source, and the cause is application-specific. The application object sends the event trigger (typically Event Active indication) to the Event Log Object.

### 5-45.1 Event Control

Events can be enabled and disabled as needed. When an event is disabled, all event triggers are ignored, meaning no log entries occur. Events can also be silenced. A silenced event continues to log event information.

**Important:** Instances of the Event Log Object are divided into the following ranges.

**Table 5-45.1 Event Log Instance ID Ranges**

Range	Meaning	Quantity
01 - 63 <sub>hex</sub>	Public	99
64 <sub>hex</sub> - C7 <sub>hex</sub>	Vendor Specific	100
C8 <sub>hex</sub> - FF <sub>hex</sub>	Reserved by CIP for future use	56
100 <sub>hex</sub> - 2FF <sub>hex</sub>	Public	512
300 <sub>hex</sub> - 4FF <sub>hex</sub>	Vendor Specific	512
500 <sub>hex</sub> - FFFF <sub>hex</sub>	Reserved by CIP for future use	64,256

## 5-45.2 Class Attributes

**Table 5-45.2 Event Log Object Class Attributes**

Attribute ID	Need in Implementation	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
1 thru 2	These class attributes are optional and are described in Chapter 4 of this specification.						
3	Conditional <sup>3</sup>	Get	NV	Number of Instances	UINT	Number of object instances currently created at this class level of the device	The number of object instances at this class hierarchy level.
4 thru 7	These class attributes are optional and are described in Chapter 4 of this specification.						
32	Conditional	Set	NV	Time Format	USINT	Data type code for time and/or date	0xCD = DATE <sup>1</sup> 0xCE = TOD <sup>1</sup> 0xCF = DATE AND TIME <sup>1</sup> 0xD6 = FTIME <sup>2</sup> 0xD7 = LTIME <sup>2</sup> 0xD8 = ITIME <sup>2</sup> 0xDB = TIME <sup>2</sup> (Default)
33	Conditional	Set	NV	Present Time	Dependant on Attribute #32		Default time value = zero for all data types
34	Optional	Get	NV	Instance List	ARRAY of STRUCT of	List of all Instance numbers and names	
				Instance number	UINT	Number of Event Log Object instance	
				Instance name	STRINGI	Length and name of Event Log Object instance	

<sup>1</sup> These data types result in a zero value on power-up/reset unless special hardware exists

<sup>2</sup> The signed time data types shall not “roll over” into negative values; these values shall “roll over” to zero

<sup>3</sup> This attribute is required when attribute #34 is implemented.

### 5-45.2.1 Time Format

Defines the time format used for logging events. All Event Log Object instances shall use the same time format.

If time stamping is implemented, this attribute is required. If time stamping is not supported, this attribute shall not be implemented.

The time formats TIME OF DAY, DATE and DATE AND TIME, if implemented, require the device to continue advancing time across power cycling/reset actions.

Not all time formats are required to be supported. The time format “TIME” is required. If the implementation only supports the default time format “TIME”, this attribute is not required to be settable.

### 5-45.2.2 Present Time

This attribute exposes a real time counter whose unit of measure is determined by the **Time Format** attribute. If time stamping is enabled in the Data Log, the value in this attribute is the source of the time stamp information.

If time stamping is supported, this attribute is required. If time stamping is not supported, this attribute shall not be implemented.

If the Set service is implemented for this attribute, the entire range of values, for the time format in effect, shall be supported for the Set service.

Attribute value roll over behavior: For any of the time formats that allow a negative value, the Present Time attribute shall not roll over into negative values.

### 5-45.2.3 Instance List

The number(s) and name(s) of all instances contained in the Event Log Class.

## 5-45.3 Instance Attributes

**Table 5-45.3 Event Log Instance Attributes**

Attr ID	Need in Implem	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
1	Optional	Set <sup>3</sup>	NV	Instance Name	STRINGI	Name of this event instance	
2	Required	Get	V	State	USINT	State of this instance.	0 = Nonexistent 1 = Stopped 2 = Empty 3 = Available 4 = Full/Overwrite 5 = Full/Halted 6 – 255 = Reserved
3	Conditional <sup>1</sup>	Get	NV	Event List Size	UDINT	Number of elements in the Event List, Event Enable, Event Silenced and Event State attribute arrays.	See semantics
4	Optional	Set	NV	Event List	ARRAY of STRUCT	All event identifiers defined for this instance.	Event Identifier format is defined by attribute 24.
5	Optional	Set	NV	Event Enable	ARRAY of BOOL	Enables each event	0 = Disabled 1 = Enabled Default = Enabled
6	Optional	Set	NV	Event Silenced	ARRAY of BOOL	Inhibits selected Event from continuing to signal it's presence in the Event	0 = Not silenced 1 = Silenced

**Event Log Object, Class Code: 41<sub>Hex</sub>**

Attr ID	Need in Implem	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
						Log as an active event, regardless of it's present condition (Active/Inactive, Logged/Not Logged)	Default = Not silenced
7	Optional	Get	V	Event State	ARRAY of BYTE	Identifies the current state of each event	<u>Bit 0 (See attribute #5)</u> 0 = Disabled 1 = Enabled <u>Bit 1</u> 0 = Inactive 1 = Active <u>Bit 2</u> 0 = UnAcked 1 = Acked <u>Bit 3</u> 0 = Not Logged 1 = Logged <u>Bit 4 (See attribute #6)</u> 0 = Not Silenced 1 = Silenced <u>Bits 5_7</u> (reserved) Default = 0x0B
8	Optional	Set	NV	Logged States Configuration	BYTE	Configures which event state transitions are logged	See Semantics below. Default = 0x01
9	Optional	Set	NV	Logged Data Configuration	BYTE	Configures which data is stored in the event log.	See Semantics below. Default = 0
10	Optional	Set	NV	Log Full Action	USINT	Configures the action to take when a new event is detected and the log is full.	0 = Halt (Default) 1 = Scroll 2 – 255 = Reserved
11	Optional	Set	NV	Duplicate Event Action	USINT	Configures the action to take when a duplicate event is detected.	0 = Ignore (Default) 1 = Add 2 = Overwrite 3 – 255 = Reserved
12	Required	Get	NV	Event/Data Log Maximum Size	UDINT	Maximum number of allowable entries in Event/Data Log.	
13	Required	Get	V	Event/Data Log Size	UDINT	The present number of entries in the Event/Data Log.	0 to Event Log Maximum Size
14	Required	Set	V <sup>2</sup>	Event/Data Log	ARRAY of STRUCT	List of all events that have been logged	See semantics
15	Optional	Get	V	Active Event/Data List Size <sup>1</sup>	UDINT	Number of entries in Active Event List	0 to Event List Size
16	Optional	Get	V	Active Event/Data List	ARRAY of STRUCT	List of event identifiers that are presently active	See semantics

Event Log Object, Class Code: 41<sub>Hex</sub>

Attr ID	Need in Implem	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
17	Optional	Get	V	Logged Event/Data List Size <sup>1</sup>	UDINT	Number of entries in Logged Event List	0 to Event Log Maximum Size
18	Optional	Get	V	Logged Event/Data List	ARRAY of STRUCT	List of event identifiers that are currently logged	See semantics
19	Optional	Get	V	Log Full	BOOL	TRUE if Event Log Size = Event Log Maximum Size	0= Log not full 1 = Log full
20	Optional	Get	V	Log Contains Entries	BOOL	Indicates if there are any events in the log.	0 = Log empty 1 = Log contains entries
21	Optional	Get	V	Log Overrun	BOOL	Indicates if there has been an overrun of the log.	0 = No Log Overrun 1 = Log Overrun
22	Optional	Set	V	Sequential Event/Data Access	See Attribute 9	Provides a simple mechanism to access Event/Data entry	See Semantics
23	Optional	Set	NV	Startup Behavior	USINT	Specifies the behavior of the instance after Power-up or Reset	0= Auto Active (Default) 1= Start Required 2-255 = Reserved
24	Optional	Set	NV	Event Identifier Format	USINT	Defines the format of the Event Identifier for the log entries.	See Semantics Default = 0

<sup>1</sup> This attribute indicates the size of a ‘list’ provided by another attribute and is required if that ‘list’ attribute is present.

<sup>2</sup> The logged data is optionally nonvolatile.

<sup>3</sup> If dynamic instances are possible (via the Create service) and this attribute is implemented, the SetAttributeSingle service is required.

## 5-45.4 Semantics:

### 5-45.4.1 Attribute #1 - Instance Name

Further defines the instance by permitting a string name for the event instance.

### 5-45.4.2 Attribute #2 - State

See State Transition Diagram (STD) and State Event Matrix (SEM) in section <??? determined at spec integration> below.

### 5-45.4.3 Attribute #3 - Event List Size

The number of members in the Event List, Event Enabled, Event Silenced and Event State attributes.

### 5-45.4.4 Attribute #4 - Event List

The list of unique event identifiers recorded by this object instance. This may not include all events which could be logged.

**5-45.4.5 Attribute #5 - Event Enable**

Enables or disables the event completely with regard to the Event Log Object. Default is enabled

**5-45.4.6 Attribute #6 - Event Silenced**

Disables the updating of the selected Event regardless of the state of the event, which allows the user to stop transmission of individual event indication messages.

**5-45.4.7 Attribute #7 - Event State**

Current event state for each of the events. Refer to Object Behavior section.

**5-45.4.8 Attribute #8 - Logged States Configuration**

Defines when an event log entry is recorded, based on event state changes. Allows the user to select which state changes require an event log entry, optimizing event log memory usage. More than one state change can be enabled simultaneously. Selecting no state changes disables event logging without disabling the state monitoring capability of the object.

**Table 5-45.4 Defined Values For the Logged States Configuration Attribute**

Value	Description
Bit 0	Log state transitions to Active or Inactive
Bit 1	Log state transitions to Disabled or Enabled
Bit 2	Log state change due to UserAck
Bits 3 - 7	Reserved

**5-45.4.9 Attribute #9 - Logged Data Configuration**

Defines the information to be stored in the Event/Data Log (Attribute 14). The logged data shall always begin with the Event Identifier (format defined in Attribute 24), and may be followed by optional data blocks as they appear in the table below, from low bit (Bit 0) to high bit (Bit 7). If a bit is set, the corresponding data block appears in the Event/Data Log.

**Table 5-45.5 Defined Values For the Logged Data Configuration Attribute**

Value	Description
Bit 0	Time value (data value defined by Time Format class attribute)
Bit 1	New event state (data value defined by Event State attribute)
Bit 2	Application data <sup>1</sup>
Bits 3 - 7	Reserved

<sup>1</sup> The Application data shall be in the form of a Data Segment (see Appendix C)

**5-45.4.10 Attribute #10 - Log Full Action**

Defines the action to take when a new event is detected and the event log is full. Options include scroll and halt. The scroll selection causes new event log entries to overwrite the oldest entries when the event log is full. The halt selection causes all new event log entries to be lost when the event log is full.

**Table 5-45.6 Defined Values For the Log Full Action Attribute**

Value	Description
0	Halt (stop when log is full)
1	Scroll (oldest entry is deleted, member list is shuffled down, and newest entry is last in the list)
2 – 255	Reserved

**5-45.4.11 Attribute #11 - Duplicate Event Action**

Defines the action to take when a duplicate event is detected. Options include ignore, add, and overwrite. The ignore option causes event entries of the same Event Identifier to be discarded. The overwrite option causes event entries of the same event to overwrite the previous entry of that event. The add to log option allows the same event to be placed in the log each time it occurs.

**Table 5-45.7 Defined Values For the Duplicate Event Action Attribute**

Value	Description
0	Ignore
1	Add
2	Overwrite
3 – 255	Reserved

**5-45.4.12 Attribute #12 - Event/Data Log Maximum Size**

The maximum number of allowable event/data log entries for the instance. This number may vary based on available memory and how the user has configured the event log options.

**5-45.4.13 Attribute #13 - Event/Data Log Size**

The current number of non-zero entries in an instance's event/data log.

**5-45.4.14 Attribute #14 - Event/Data Log**

This attribute is the event/data log for the instance. New events are added to the end of the log, subject to overriding behavior specified by the Log Full Action and Duplicate Event Action attributes.

**Event Log Object, Class Code: 41<sub>Hex</sub>**

Information contained in the event/data log depends upon its configuration. An entry whose value is all zero is defined to be a null entry containing no useful data. This attribute is a single dimension array of entries. The maximum size of the array is given by attribute #12 (Event/Data Log Maximum Size). The present number of entries is given by attribute #13 (Event/Data Log Size). The form of each entry is determined by the combination of the Event Identifier Format (attribute #24) in conjunction with optional data selected by the Logged Data Configuration (attribute #9). The table below shows the structure of a logged event entry.

**Table 5-45.8 Structure of Event Entry**

Field Name	Need in Entry	Where enabled	Where Format Defined
Event Identifier	Required	Always Present	Event Identifier Format (Instance Attribute 24)
Time Stamp	Optional	Logged Data Configuration (Instance Attribute 9), Bit 0	Time Format (Class Attribute 32)
Event State	Optional	Logged Data Configuration (Instance Attribute 9), Bit 1	Event State (Instance Attribute 7)
Application Data	Optional	Logged Data Configuration (Instance Attribute 9), Bit 2	Encoded Data Segment as defined in Appendix C

For example, assuming the Log Entry Form attribute is the value zero, then the first four octets of each member entry shall conform to the 32-bit object model/error response format (the Event Identifier). If any optional data is selected by the Logged Data Configuration attribute, the selected data will be present in the logged event entry following the Event Identifier. The order of the selected fields is defined by the Logged Data Configuration attribute and shown in the table above. Continuing this example, assuming the value 0x03 exists in the Logged Data Configuration attribute and class attribute #32 (Time Format) is the value 0xDB (TIME data type), the following table shows the form of each Event/Data Log member entry:

**Table 5-45.9 Event/Data Log Member Entry Form**

Byte Offset	Contains	
0	Event Identifier	8-bit Class Code
1		8-Bit Instance
2		8-Bit Error/Event Code
3		8-Bit Extended Event/Error Code
4	Time Value	Time Stamp for entry (Low Order Byte)
5		Time Stamp (next byte)
6		Time Stamp (next byte)
7		Time Stamp (High Order Byte)
8	New Event State	Event State

**5-45.4.14.1 Service Behavior:****5-45.4.14.1.1 Get\_Attribute\_Single**

Returns the entire content of the attribute. Each member of the Event Log is returned, regardless of whether the content of the member is non-null (valid). Therefore, this service always returns “Event Log Maximum Size” times each entry’s “form size” bytes.

**Table 5-45.10 Get\_Attribute\_Single Service Error Code**

Error Code	Error Name	Description
0x15	Too Much Data	The present size of the message necessary to transmit all Event Log members exceeds the messaging capacity of the node

**5-45.4.14.1.2 Set\_Attribute\_Single**

The Set\_Attribute\_Single service shall not be implemented

**Table 5-45.11 Set\_Attribute\_Single Service Error Code**

Error Code	Error Name	Description
0x08	Service Not Supported	The Set service is not supported for this attribute

**5-45.4.14.1.3 Get\_Member**

Returns the content of the specified member within the Event/Data Log

**Table 5-45.12 Get\_Member Service Error Codes**

Error Code	Error Name	Description
0x28	Invalid Member ID	Member number zero was selected
0x08	Service Not Supported	This implementation of the Event Log object does not support the Get Member service

**5-45.4.14.1.4 Set\_Member****Table 5-45.13 Set\_Member Service Error Codes**

Error Code	Error Name	Description
0x28	Invalid Member ID	The specified member does not presently exist
0x15	Too Much Data	The service data of the request contained too much data for the form of the Event Log
0x13	Not Enough Data	The service data of the request did not contain enough data for the form of the Event Log
0x0C	Object State Conflict	The state of the instance is any of: Stopped Halted or the specified member is valid and can not be overwritten
0x09	Invalid Attribute Value	The value of the data within the request was rejected by the Event Log
0x08	Service Not Supported	This implementation of the Event Log object does not support the Set Member service

**5-45.4.14.1.5 Remove\_Member**

Returns the content of the specified member within the Event/Data Log and then the member is removed from the Log.

**Table 5-45.14 Remove\_Member Service Error Codes**

Error Code	Error Name	Description
0x28	Invalid Member ID	The specified member does not exist
0x08	Service Not Supported	This implementation of the Event Log object does not support the Remove Member service

**5-45.4.14.1.6 Insert\_Member**

Inserts the content of the service data into the Event/Data Log at the specified member position.

**Table 5-45.15 Insert\_Member Service Error Codes**

Error Code	Error Name	Description
0x28	Invalid Member ID	The specified member does not exist
0x15	Too Much Data	The service data of the request contained too much data for the form of the Event Log
0x13	Not Enough Data	The service data of the request did not contain enough data for the form of the Event Log
0x0C	Object State Conflict	The state of the instance is any of: Stopped Halted or the specified member is valid and can not be moved in the member list
0x09	Invalid Attribute Value	The value of the data within the request was rejected by the Event Log
0x08	Service Not Supported	This implementation of the Event Log object does not support the Insert Member service

**5-45.4.15 Attribute #15 - Active Event List Size**

The current number of entries in the Active Event List.

**5-45.4.16 Attribute #16 - Active Event List**

A list of all currently active events related to the instance from which the list is obtained.

**5-45.4.17 Attribute #17 - Logged Event List Size**

The current number of entries in the Logged Event List. This list shall be the same size as the Event List.

**5-45.4.18 Attribute #18 - Logged Event List**

A list of all currently logged events related to the instance from which the list is obtained.

**5-45.4.19 Attribute #19 - Log Full**

A flag which indicates if the Event Log is full. TRUE if Event Log Size = Event Log Maximum Size, and intended primarily for dynamic log management.

**5-45.4.20 Attribute #20 - Log Not Empty**

A flag which indicates that the Event Log is not empty (ie. at least one entry is present). This value is TRUE when the Event Log Size is not equal to zero.

**5-45.4.21 Attribute #21 - Log Overrun**

A flag which indicates that an overrun has occurred. This value is reset to zero (FALSE) when the attribute is read.

**5-45.4.22 Attribute #22 – Sequential Event/Data Access**

The Sequential Event/Data Access attribute provides the means for simple sequential access to members of the Event/Data Log. The form of the data in this attribute is identical to the form of the data contained within individual member(s) of the Event/Data Log (attribute #14). The behavior of services to the Sequential Event/Data Access attribute is as follows:

**Table 5-45.16 Sequential Event/Data Access Behavior**

Sequential Event/Data Access (attribute #22)		Results in Service to Event/Data Log (attribute #14)		
Service Code	Service Name	Service Code	Service Name	Behavior
0x0E	Get_Attribute_Single	0x1B	Remove_Member	<p>The Get_Attribute_Single service, when directed to the Sequential Event/Data Access attribute, causes a Remove_Member service to be directed to the Event/Data Log attribute of the Event Log object instance. The internally generated Remove_Member service will always attempt to remove member one of the Event/Data Log attribute.</p> <p>If the Event/Data log is empty, this service shall return no data.</p> <p>If the Sequential Event/Data Access attribute is implemented, support for the Remove_Member service is required.</p>
0x10	Set_Attribute_Single	0x1A	Insert_Member	<p>The Set_Attribute_Single service, when directed to the Sequential Event/Data Access attribute, causes an Insert_Member service to be directed to the Event/Data Log attribute of the Event Log object instance. The internally generated Insert_Member service will always attempt to insert the last member of the Event/Data Log attribute.</p> <p>If the Set_Attribute_Single service is implemented for this attribute support for the Insert_Member service is required.</p>

**5-45.4.23 Attribute #23 - Startup Behavior**

Defines the behavior of the instance state machine at startup and reset.

**Table 5-45.17 Startup Behavior Attribute Values**

Value	Description
0	Auto Active
1	Start Required
2 – 255	Reserved

**5-45.4.24 Attribute #24 - Event Identifier Format**

Defines the format of the Event Identifier.

**Table 5-45.18 Values for Attribute #24**

Value	Description
0	32-bit object model/error format
1	48-bit object model/error format
2	64-bit object model/error format
3	16-bit unique identifier value
4 – 254	Reserved
255	Product specific – fixed form

**Table 5-45.19 32-Bit Object Model / Error Format**

Byte Offset	Data Type	Description
0	USINT	8-bit object class identification
1	USINT	8-bit object instance identification
2	USINT	8-bit Error/Event code
3	USINT	8-bit Extended Error Code

**Table 5-45.20 48-Bit Object Model / Error Format**

Byte Offset	Data Type	Description
0	UINT	16-bit object class identification
2	USINT	8-bit object instance identification
3	USINT	8-bit Error/Event code
4	UINT	16-bit Extended Error Code

**Table 5-45.21 64-Bit Object Model / Error Format**

Byte Offset	Data Type	Description
0	UINT	16-bit object class identification
2	UINT	16-bit object instance identification
4	UINT	16-bit Error/Event code
6	UINT	16-bit Extended Error Code

**Table 5-45.22 16-Bit Unique Identifier Value**

Byte Offset	Data Type	Description
0	UINT	16-bit unique identifier

## 5-45.5 Common Services

**Table 5-45.23 Common Services for the Event Log Object**

Service Code	Need In Implementation		Service Name	Service Description
	Class	Instance		
05 <sub>hex</sub>	N/A	Required	Reset	Provides one of the types of reset for the object as described below.
06 <sub>hex</sub>	N/A	Conditional <sup>1</sup>	Start	Transitions the object from the Configuring state to the Active state, which allows event logging to commence.
07 <sub>hex</sub>	N/A	Conditional <sup>1</sup>	Stop	Transitions the object from the Active state to the Configuring state. This service causes the event list and status attributes to be cleared and allows for the setting of some attributes which are get-only in the Active state.
0E <sub>hex</sub>	Optional <sup>2</sup>	Required	Get_Attribute_Single	Returns the contents of the specified attribute.
10 <sub>hex</sub>	Optional	Optional	Set_Attribute_Single	Sets the contents of the specified attribute.
18 <sub>hex</sub>	Optional	Optional	Get_Member	Returns the contents of a member of the specified attribute.
19 <sub>hex</sub>	Optional	Optional	Set_Member	Modifies a member of the specified attribute.
1A <sub>hex</sub>	N/A	Optional	Insert_Member	Adds a member to one of the list attributes.
1B <sub>hex</sub>	N/A	Optional	Remove_Member	Removes a member from one of the list attributes.

1 The Delete, Start and Stop services are required if the Create Service is supported.

2 The Get\_Attribute\_Single service is required if any attributes are implemented.

### 5-45.5.1 Reset Service

The Reset service has the following object specific parameter:

**Table 5-45.24 Reset Service Object Specific Parameters**

Name	Data Type	Description of Parameter	Semantics of Values
Reset Type	USINT	Type of reset requested.	See table below.

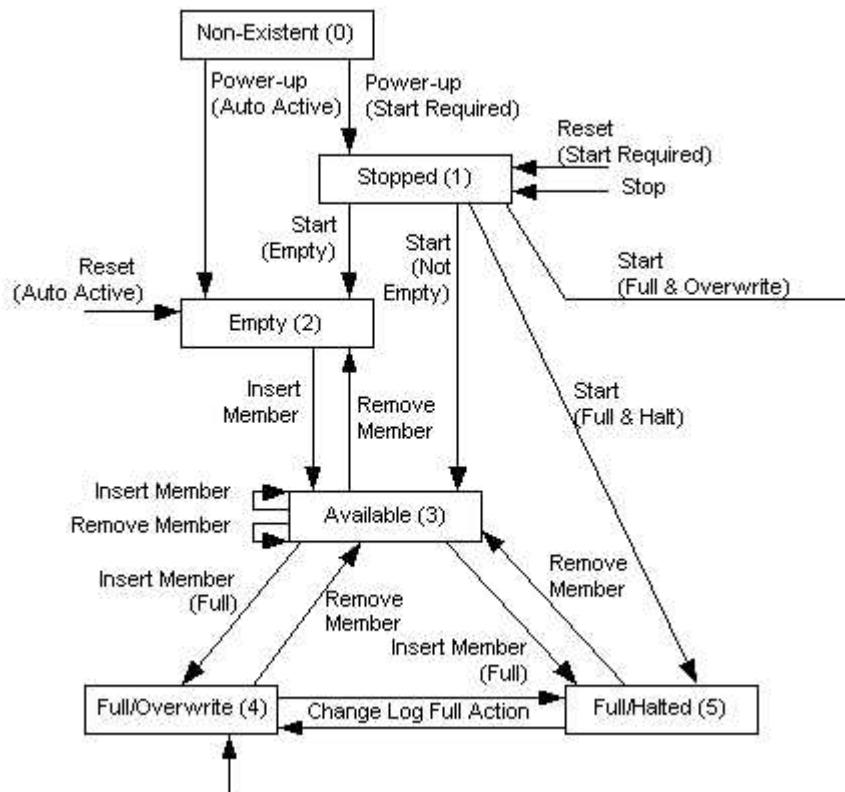
The parameter Reset Type for the Reset service has the following value specification:

**Table 5-45.25 Reset Type Value Descriptions**

Value	Description of Value
0	Emulate as closely as possible cycling power, returning all attributes to the power up values. This is the default if the parameter is omitted.
1	Return as closely as possible to the out-of-box configuration, then emulate cycling power as closely as possible.
2 – 255	Reserved.

## 5-45.6 Event/Data Log Instance Behavior

**Figure 5-45.26 Event/Data Log Instance State Transition Diagram**



Event Log Object, Class Code: 41<sub>Hex</sub>

Table 5-45.27 Event/Data Log Object State Event Matrix

Event	Event/Data Log Object Instance State Event Matrix									
	Non-Existent (0)	Stopped (1)	Empty (2)	Available (3)	Full Overwrite (4)	Full Halted (5)				
Power Up	<i>If</i> set to “Power up active” transition to Empty <i>Else</i> transition to Stopped.	Not Applicable	Not Applicable	Not Applicable	Not Applicable	Not Applicable				
Create	Transition to Stopped	Not Applicable	Not Applicable	Not Applicable	Not Applicable	Not Applicable				
Delete	Not Applicable	Delete all resources and transition to Non-Existent.	Return Error (Object State Conflict)	Return Error (Object State Conflict)	Return Error (Object State Conflict)	Return Error (Object State Conflict)				
Start (Not supported)	Not Applicable	Return Error (Service not supported)	Return Error (Service not supported)	Return Error (Service not supported)	Return Error (Service not supported)	Return Error (Service not supported)				
Start (Supported)	Not Applicable	Transition to Available	Return Error (Object Already in State/Mode)							
Stop (Not supported)	Not Applicable	Return Error (Service not supported)	Return Error (Service not supported)	Return Error (Service not supported)	Return Error (Service not supported)	Return Error (Service not supported)				
Stop (Supported)	Not Applicable	Return Error (Object Already in State/Mode)	Clear the Event Log, Active Event List, Logged Event List. Set Log Full, Log Not Empty, and Log Overrun attributes to zero. Transition to Stopped.							
Reset (Type = 0 or 1)	Not Applicable	Perform Reset as defined by service.	Perform Reset as defined by service.							
Get Attribute	Not Applicable	Validate/service the request. Return response.								
Set Attribute	Not Applicable	Validate/service the request. Return response.								
SetAttribute	Not Applicable	Validate/service the request. Return response.								
to “Log Full Action” attribute					If attribute value after service is zero (Halt) – Transition to Full Halted	If attribute value after service is one (Scroll) – Transition to Full Scroll				
Insert Member (internal)	Not Applicable	Ignore	<i>If</i> event Not Enabled discard <i>Else</i> process event based on Logged Data, Log Full Action, and Duplicate Event Action attributes. Update Log Full, Log Not Empty, and Log Overrun attributes accordingly.			Ignore				
			Transition to Available state	If Log is now full and Log Full Action is “Halt” – Transition to “Full Halted”  If Log is now full and Log Full Action is “Scroll” – Transition to “Full Scroll”						

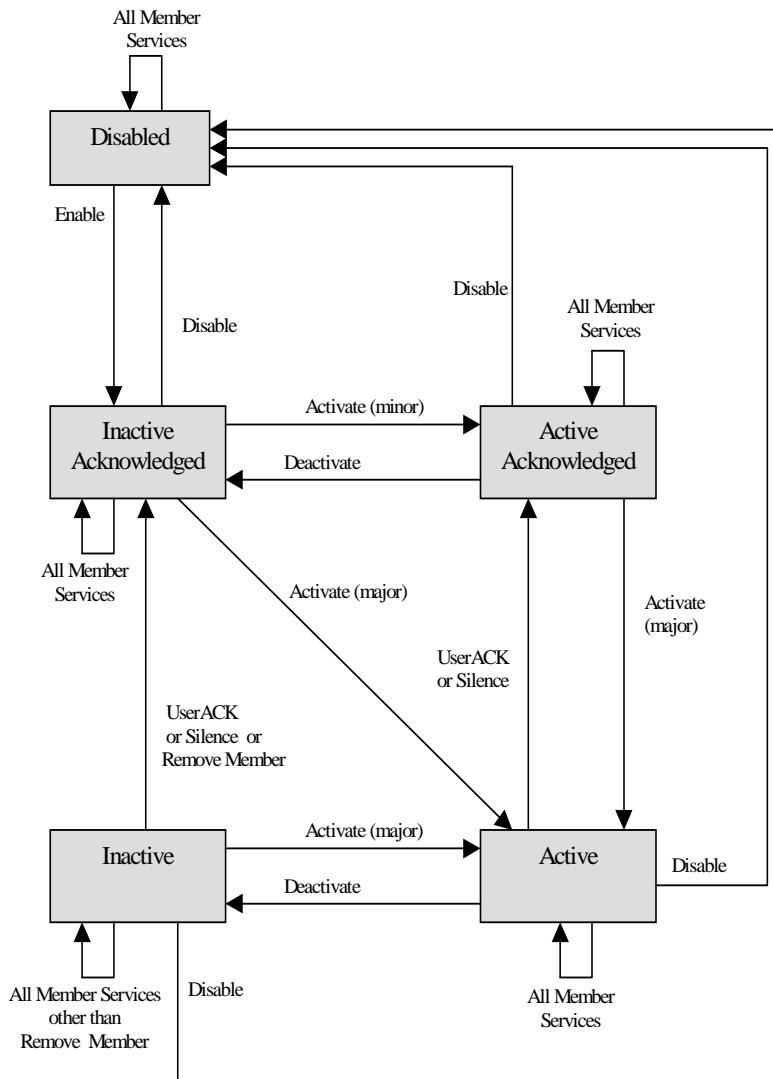
Event Log Object, Class Code: 41<sub>Hex</sub>

Event	Event/Data Log Object Instance State Event Matrix					
	Non-Existent (0)	Stopped (1)	Empty (2)	Available (3)	Full Overwrite (4)	Full Halted (5)
Insert Member (via explicit service or the Sequential Event/Data Access attribute)	Not Applicable	Return error (Object State Conflict)	<p>If event Not Enabled discard</p> <p>Else process event based on Logged Data, Log Full Action, and Duplicate Event Action attributes. Update Log Full, Log Not Empty, and Log Overrun attributes accordingly.</p>			Return error (Object State Conflict)
			Transition to Available state	If Log is now full and Log Full Action is “Halt” – Transition to “Full Halted”  If Log is now full and Log Full Action is “Scroll” – Transition to “Full Scroll”		
Remove Member	Not Applicable	Return error (Object State Conflict)	Return error (Object State Conflict)	Remove member at specified index. Update Log Full and Log Not Empty attributes accordingly.		
				If last entry removed, Transition to Empty	Transition to Available	Transition to Available
Get Member	Not Applicable	Validate/service the request. Return response.				

**5-45.6.1 Event Behavior**

The Event Log object defines optional states and state transitions for events. This behavior definition does not define what physically causes the transition triggers. For example, UserACK/Silence could be a pushbutton on one device, and may require an HMI interactive sequence on another. The behavior of an event is illustrated in the State Transition Diagram and State Event Matrix below.

Figure 5-45.28 Event State Transition Diagram



Event Log Object, Class Code: 41<sub>Hex</sub>

Table 5-45.29 State Event Matrix for Events

Event Trigger	State				
	Disabled	Inactive-Acked	Active-Acked	Inactive	Active
Disable	Ignore	Transition to Disabled	If not Silenced, Event State = Inactive-Acked Transition to Disabled	If not Silenced, Event State = Inactive-Acked Transition to Disabled	If not Silenced, Event State = Inactive-Acked Transition to Disabled
Enable	Transition to Inactive-Acked	Ignore	Ignore	Ignore	Ignore
Silence	Ignore	Ignore	Ignore	Event State = Inactive-Acked Transition to Inactive-Acked	Event State = Active-Acked Transition to Active-Acked
Unsilence	Ignore	Event State = Inactive-Acked	Event State = Active-Acked	Not applicable	Not applicable
Activate (major)	Ignore	If not silenced, Event State = Active Log event (if enabled) If silenced, transition to Active-Acked If not silenced, transition to Active	If not silenced, Event State = Active Log event (if enabled) If not silenced, transition to Active	If not silenced, Event State = Active Log event (if enabled) If silenced, transition to Active-Acked If not silenced, transition to Active	Not applicable
Activate (minor)	Ignore	If not silenced, Event State = Active-Acked Log event (if enabled) Transition to Active-Acked	Ignore	Not applicable	Not applicable
Deactivate	Ignore	Ignore	If not silenced, Event State = Inactive-Acked Transition to Inactive-Acked	Ignore	If not silenced, Event State = Inactive If not silenced, transition to Inactive. If silenced, transition to Inactive-Acked.
UserACK	Ignore	Ignore	Ignore	If not silenced, Event State = Inactive-Acked Transition to Inactive-Acked	If not silenced, Event State = Active-Acked Transition to Active-Acked
All Member Services (except Remove_Member)	Service the request and return the appropriate response.	Service the request and return the appropriate response.	Service the request and return the appropriate response.	Service the request and return the appropriate response.	Service the request and return the appropriate response.
Remove_Member	Service the request and return the appropriate response.	Service the request and return the appropriate response.	Return Object State Conflict error (0C hex)	Service the re-quest and return the appropriate response. Transition to Inactive-Acked	Return Object State Conflict error (0C hex)

## **5-46 Motion Axis Object**

**Class Code: 42<sub>hex</sub>**

### **5-46.1 Introduction**

This object defines the behavior of a CIP object called the Motion Axis Object. This object is the main functional component of CIP Motion device profiles defined as part of the CIP Motion extensions to CIP, e.g., the CIP Motion Drive Device Profile. Instances of this object are required to support motion control using devices compliant with the CIP Motion Device Profile(s) (“CIP Motion Device(s)”) over a CIP Network. The object may be adapted to **any CIP network**. It is generally contemplated that EtherNet/IP would be the network designer’s technology of choice for high performance, synchronized multi-axis control.

In this object definition, we use these terms interchangeably: “device”, “motion device” & “drive”.

### **5-46.2 Organization**

The Motion Axis Object covers the behavior of various motion control system devices that includes feedback devices and drive devices. For drive devices, the Motion Axis Object covers a wide range of drive types from simple variable frequency (V/Hz) drives, to servo drives. Many drive products can be configured to operate in different modes depending on the specific application requirement. The attributes of the Motion Axis Object are therefore organized to address this broad range of functionality and the framework for this organization is described in the following section.

#### **5-46.2.1 Control Modes**

This object is organized around the general philosophy that position control is the highest order of dynamic motion control. That is, position control implies velocity control, and velocity control implies acceleration control. Acceleration is related to torque or force by the inertia or mass of the load, respectively, acceleration control implies torque control. And finally, since motor torque or force is generally related to motor current by a torque or force constant, respectively, torque control implies current control. The torque or force constant can be a function of the motor magnets as in a Permanent Magnet motor, or the induced flux of an Induction motor.

Since acceleration, torque/force, and current are generally related by a constant, these terms are sometimes used interchangeably in the industry. For example, a torque control loop rather than a current control loop. This object attempts to differentiate between these control properties were applicable. This is particularly useful when the relationship between them is not static, such as when inertia/mass changes with position or time, or when the torque/force constant changes due to temperature change or motor flux variation.

#### **5-46.2.2 Control Methods**

Within this basic control paradigm, there is latitude for different control methods, both closed loop and open loop. By closed loop, it is generally implied that there is a feedback signal that is used to drive the actual dynamics of the motor to match the commanded dynamics by servo action. In most cases there is a literal feedback device to provide this signal, and in some cases the signal is derived from the motor excitation (i.e. sensorless operation). By open loop, it is implied that there is no application of feedback to force the actual dynamics to match the commanded dynamics.

### 5-46.2.3 Control Nomenclature

Typically rotary applications speak of Torque and Inertia while linear applications speak of Force and Mass. For the purposes of this document, when we refer to rotary nomenclature, the defined behavior can generally be applied to linear applications by substituting the terms, force for torque and mass for inertia. With that understanding, we use torque rather than force in the diagrams below without loss of generality.

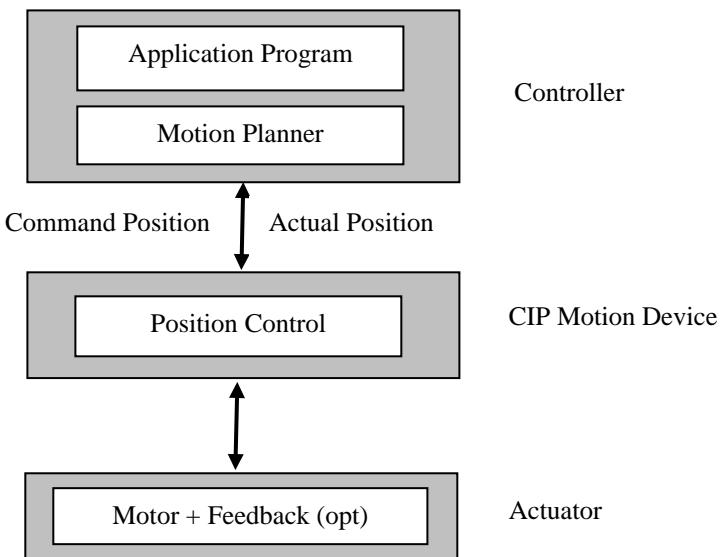
### 5-46.2.4 Position Control

In position control application mode either the application control program (command execution function) or the motion planner (move trajectory control function) provide a set-point value to a CIP Motion Device via the cyclic data connection. The position control method can be either open loop or closed loop.

#### 5-46.2.4.1 Open Loop Position Control

A device configured for open loop position control applies to a class of drive devices called stepper drives. This type of drive is illustrated below.

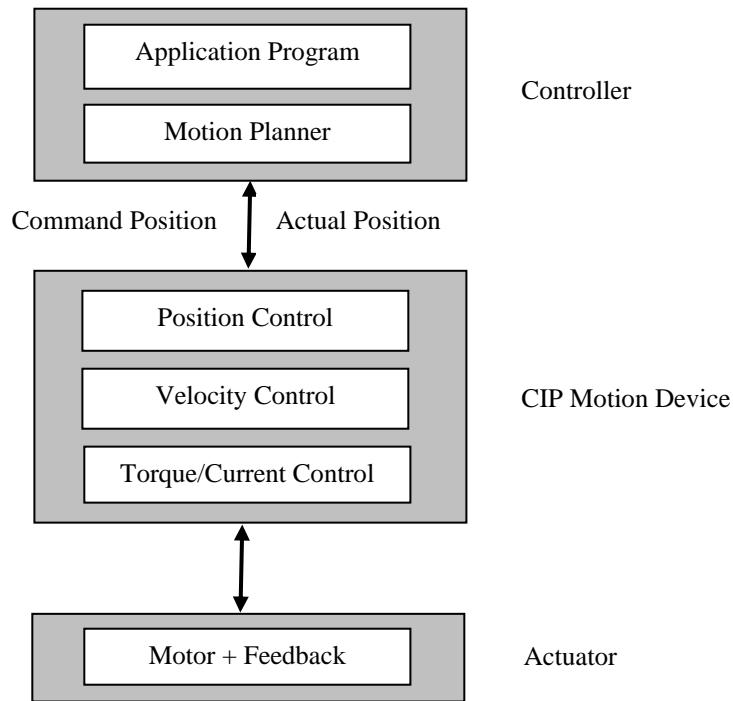
**Figure 5-46.1 Open Loop Position Control**



A feedback device for this configuration is optional. In the absence of a feedback device, actual position can be estimated by the drive and returned to the controller.

#### 5-46.2.4.2 Closed Loop Position Control

A motion control device configured for closed loop position control is typically referred to as position loop drive or position servo drive. A position servo drive implies an inner velocity and torque control loop as shown below. The presence of the torque/current control loop sometimes results in this kind of drive being referred to as a vector drive.

**Figure 5-46.2 Closed Loop Position Control**

A feedback device for this configuration is generally required to achieve good positioning accuracy. The feedback device may also be used to return Actual Velocity and Actual Acceleration data to the controller via the cyclic data connection.

In addition to Command Position, the controller can pass Command Velocity and Command Acceleration for the purposes of forward control.

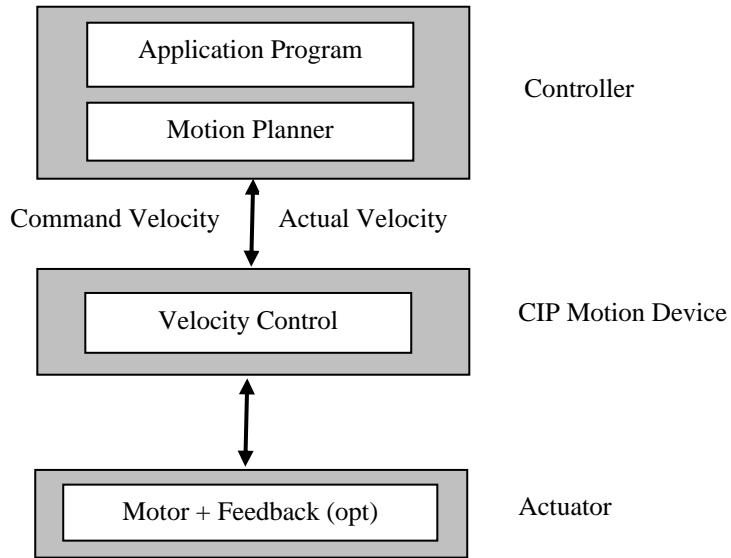
### 5-46.2.5 Velocity Control

In velocity control application mode the application control program and motion planner provide a set-point value to a CIP Motion Device via the cyclic data connection. The velocity control method can be either open loop or closed loop.

#### 5-46.2.5.1 Open Loop Velocity Control

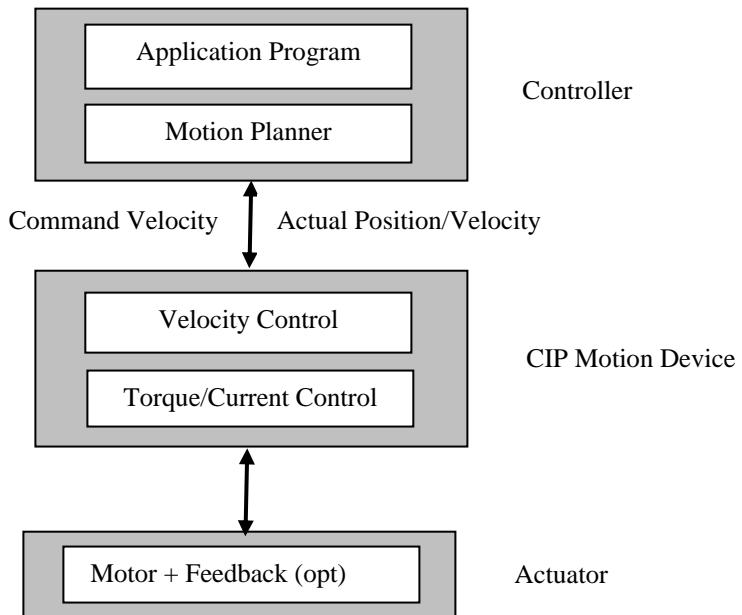
A motion control device configured for open loop velocity control is typically referred to as Variable Frequency, or V/Hz, or VFD, drive. This type of drive is illustrated below.

A feedback device for this configuration is optional. In the absence of a feedback device, actual velocity can be estimated by the drive and returned to the controller.

**Figure 5-46.3 Open Loop Velocity Control**

### 5-46.2.5.2 Closed Loop Velocity Control

A motion control device configured for closed loop velocity control is typically referred to as velocity loop drive or velocity servo drive. A closed loop velocity control drive implies an inner torque/current control loop and therefore is sometimes referred to as a vector drive.

**Figure 5-46.4 Closed Loop Velocity Control**

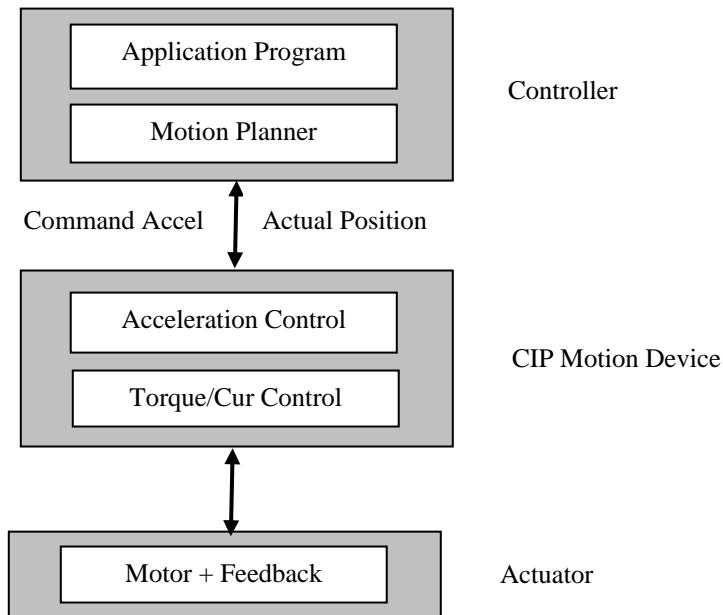
A feedback device for the velocity loop drive configuration is optional. Tighter speed regulation may be achieved when using a feedback device, particularly at low speed. When the feedback device is included it may be used to return actual position, velocity, and acceleration data to the controller via the cyclic data connection. When the feedback device is not included, only estimated velocity can typically be returned to the controller.

In addition to Command Velocity, the controller can also pass Command Acceleration for the purposes of forward control.

#### 5-46.2.6 Acceleration Control

While not a mainstream control mode in the industry, acceleration control mode is included here to complete the dynamic progression from velocity control to torque control and because the Motion Axis Object can support an Acceleration Command, potentially derived from the controller's motion planner. In the acceleration control mode, the application control program and motion planner provide acceleration set-point values to a CIP Motion Device via the cyclic data connection. The drive converts the acceleration set-point into a torque command using the estimated system inertia. Acceleration control works in concert with the inner torque/current control loop as shown below.

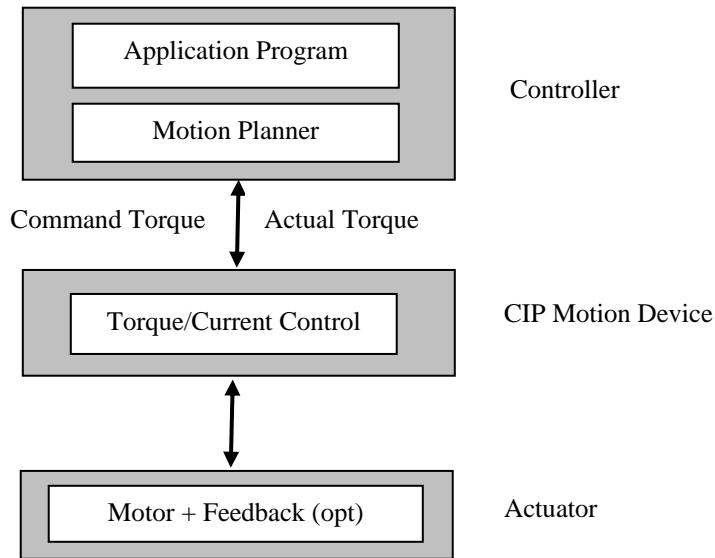
**Figure 5-46.5 Acceleration Control**



A feedback device for the acceleration control configuration is mandatory and may be used to return actual position, velocity, and acceleration data to the controller via the cyclic data connection.

#### 5-46.2.7 Torque Control

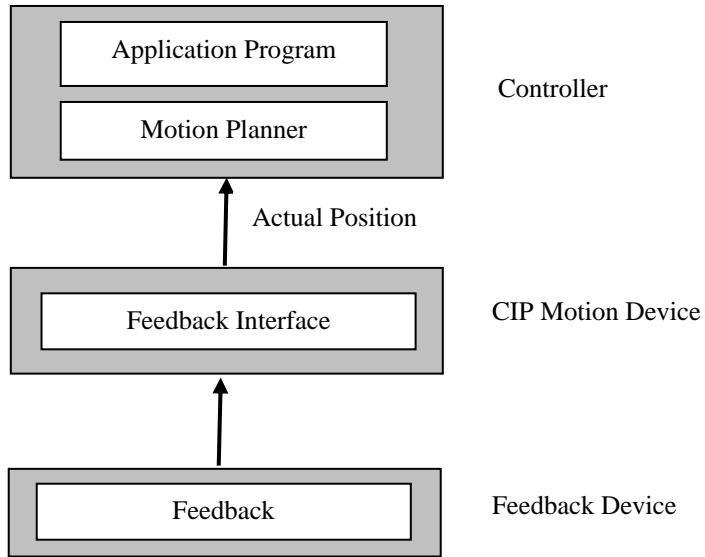
In torque control application mode, the application control program or the motion planner provide torque set-point values to the device via the cyclic data connection. Because motor current and motor torque are generally related by a torque constant, K<sub>t</sub>, torque control is often synonymous with current control.

**Figure 5-46.6 Torque Control**

A position feedback device for this control mode is optional. If a feedback device is present it may be used to return actual position, velocity, and acceleration data to the controller via the cyclic data connection.

#### 5-46.2.8 No Control

The Motion Axis Object supports a “No Control” application mode. This configuration also referred to as “feedback only” or “master feedback”, can be of value when a particular feedback channel in the CIP Motion Device is to serve simply as a master feedback source to the rest of the control system. This application mode can also be applied to CIP Motion Feedback devices. In this “No Control” configuration, no set-point value is supplied to the CIP Motion Device via the cyclic data connection, but actual position, velocity, and acceleration can be supplied by the device to the controller via the cyclic data channel. The No Control drive configuration is illustrated in the diagram below.

**Figure 5-46.7 No Control**

### 5-46.3 Drive Control Categories

Based on the above variations in Control Mode and Control Method we can derive some basic Device Control Codes around which we can organize the many attributes of the Motion Axis Object. Device Control Codes can be designated using a single letter identifier that can be used to determine what object attributes are required for implementation of a given CIP Motion Device. The list of Device Control Codes is as follows:

**Control Modes:**

- P – Position Control
- V – Velocity Control
- T – Torque Control
- N – No Control (Feedback Only)

**Control Methods:**

- F – Frequency Controlled Drives (Open Loop V/Hz or VFD)
- C – Closed Loop Vector Controlled Drives

Using combinations of these letters we can designate a specific class of CIP Motion devices for the purposes of identifying applicable attributes. For example, F would indicate a variable frequency drive axis, “V” would refer to velocity controlled drive axis, either open loop or closed loop, and “N” would refer to an axis with feedback only functionality.

### 5-46.4 Required vs. Optional in Implementation

In the sections that follow, attributes and services are defined as Required or Optional in the implementation of the Motion Axis Object. Required attributes and services must be supported in the implementation of the object. Optional attributes and services may or may not be supported in the implementation and are left to the discretion of the vendor.

**Motion Axis Object, Class Code: 42<sub>hex</sub>**

The determination of whether a given attribute or service is Required or Optional often depends on the associated Device Control Code as defined above. If an attribute or service is marked as Required for a given Device Control Code then a device implementation must support that attribute or service if it is intended to operate in that mode. For example an attribute marked as required for Device Control Code “V” must be supported by any CIP Motion Device that supports Velocity control mode.

In some cases an attribute or service may not even be applicable to a Device Control Code. This situation is implied when the attribute is defined as neither Required nor Optional.

The following table provides a convenient list of all the Instance Attributes of this object and identifies whether the attribute is Required or Optional in the implementation based on the above Device Control Code.

Note: If there are any discrepancies between this table and the actual attribute descriptions, the attribute descriptions prevail.

Note: The definition of the abbreviations used in the Conditional Implementation column of the following table can be found in section 5-46.10.

**Table 5-46.1 Instance Attribute Implementation vs. Drive Control Category Code**

Instance Attribute			Implementation by Drive Category Code					
Attr. ID	Access Rule	Attribute Name	N	F	C - PI Vector Ctrl			Conditional
				V	P	V	T	Implementation
80	Set*	Control Mode	-	R	R	R	R	
81	Set	Control Method	-	R	R	R	R	
1310	Get	Motor Catalog Number	-	O	O	O	O	
1311	Get	Motor Serial Number	-	O	O	O	O	
1312	Get	Motor Date Code	-	O	O	O	O	
1313	Set	Motor Data Source	-	R	R	R	R	
1314	Set	Motor Device Code	-	R	R	R	R	
1315	Set	Motor Type	-	R	R	R	R	
1316	Set	Motor Unit	-	O	O	O	O	
1317	Set	Motor Polarity	-	R	R	R	R	
1318	Set	Motor Rated Voltage	-	R	R	R	R	
1319	Set	Motor Rated Continuous Current	-	R	R	R	R	
1320	Set	Motor Rated Peak Current	-	C	C	C	C	R - PM; O - IM
1321	Set	Motor Rated Output Power	-	C	C	C	C	R - PM; O - IM
1322	Set	Motor Overload Limit	-	O	O	O	O	
1323	Set	Motor Integral Thermal Switch	-	O	O	O	O	
1324	Set	Motor Max Winding Temperature	-	O	O	O	O	
1325	Set	Motor Winding-to-Ambient Thermal Capacitance	-	O	O	O	O	
1326	Set	Motor Winding-to-Ambient Thermal Resistance	-	O	O	O	O	
1327	Set	PM Motor Resistance	-	R	R	R	R	PM Motor only
1328	Set	PM Motor Inductance	-	R	R	R	R	PM Motor only
1329	Set	Rotary Motor Poles	-	R	R	R	R	Rotary Motor only
1330	Set	Rotary Motor Inertia	-	O	O	O	O	Rotary Motor only
1331	Set	Rotary Motor Rated Speed	-	R	R	R	R	Rotary Motor only

**Motion Axis Object, Class Code: 42<sub>hex</sub>**

Instance Attribute			Implementation by Drive Category Code					
Attr. ID	Access Rule	Attribute Name	N	F	C - PI Vector Ctrl			Conditional
			V	P	V	T	Implementation	
1332	Set	Rotary Motor Max Speed	-	O	O	O	O	Rotary Motor only
1333	Set	Rotary Motor Damping Coefficient	-	O	O	O	O	Rotary Motor only
1334	Set	Linear Motor Pole Pitch	-	R	R	R	R	Linear Motor only
1335	Set	Linear Motor Rated Speed	-	R	R	R	R	Linear Motor only
1336	Set	Linear Motor Moving Mass	-	O	O	O	O	Linear Motor only
1337	Set	Linear Motor Max Speed	-	O	O	O	O	Linear Motor only
1338	Set	Linear Motor Damping Coefficient	-	O	O	O	O	Linear Motor only
1339	Set	PM Motor Rated Torque	-	R	R	R	R	Rotary PM Motor only
1340	Set	PM Motor Torque Constant	-	R	R	R	R	Rotary PM Motor only
1341	Set	PM Motor Rotary Voltage Constant	-	R	R	R	R	Rotary PM Motor only
1342	Set	PM Motor Rated Force	-	R	R	R	R	Linear PM Motor only
1343	Set	PM Motor Force Constant	-	R	R	R	R	Linear PM Motor only
1344	Set	PM Motor Linear Voltage Constant	-	R	R	R	R	Linear PM Motor only
1345	Set	Induction Motor Rated Frequency	-	R	R	R	R	Induction Motor only
1346	Set	Induction Motor Flux Current	-	R	R	R	R	Induction Motor only
1347	Set	Induction Motor Stator Resistance	-	R	R	R	R	Induction Motor only
1348	Set	Induction Motor Stator Leakage X	-	R	R	R	R	Induction Motor only
1349	Set	Induction Motor Magnetization X	-	R	R	R	R	Induction Motor only
1350	Set	Induction Motor Rotor Resistance	-	R	R	R	R	Induction Motor only
1351	Set	Induction Motor Rotor Leakage X	-	R	R	R	R	Induction Motor only
1400 + o	Get	Feedback n Catalog Number	-	O	O	O	O	
1401 + o	Get	Feedback n Serial Number	-	O	O	O	O	
1402 + o	Get	Feedback n Position	R	-	R	R	O	
1403 + o	Get	Feedback n Velocity	R	-	R	R	O	
1404 + o	Get	Feedback n Acceleration	R	-	R	R	O	
90	Set*	Feedback Data Set	R	R	R	R	R	
82	Set*	Feedback Configuration	R	R	R	R	R	
83	Set*	Feedback Master Select	R	-	R	R	R	
84	Set	Feedback 1/2 Count Ratio	-	-	O	-	-	
1411 + o	Set	Feedback n Unit	O	-	O	O	O	
1412 + o	Set	Feedback n Port Select	O	-	O	O	O	
1413 + o	Set	Feedback n Type	R	-	R	R	R	
1414 + o	Set	Feedback n Polarity	O	-	O	O	O	
1415 + o	Set	Feedback n Mode	R	-	R	R	R	
1416 + o	Set	Feedback n Cycle Resolution	R	-	R	R	R	TT, SC, HI, E21, RS
1417 + o	Set	Feedback n Cycle Interpolation	R	-	R	R	R	TT, SC, HI, E21, RS
1420 + o	Set	Feedback n Data Length	R	-	R	R	R	TP, SS, HI, E21, E22
1421 + o	Set	Feedback n Data Code	O	-	O	O	O	TP, SS
1422 + o	Set	Feedback n Resolver Transformer Ratio	O	-	O	O	O	RS
1423 + o	Set	Feedback n Resolver Excitation Voltage	O	-	O	O	O	RS

Motion Axis Object, Class Code: 42<sub>hex</sub>

Instance Attribute			Implementation by Drive Category Code					
Attr. ID	Access Rule	Attribute Name	N	F	C - PI Vector Ctrl			Conditional
			V	P	V	T	Implementation	
1424 + o	Set	Feedback n Resolver Excitation Frequency	O	-	O	O	O	RS
1425 + o	Set	Feedback n Resolver Cable Balance	O	-	O	O	O	RS
1426 + o	Set	Feedback n LDT Type	R	-	R	R	R	LT
1427 + o	Set	Feedback n LDT Recirculations	R	-	R	R	R	LT
1428 + o	Set	Feedback n Commutation Offset	R	-	R	R	R	HS, TP, HI, E21, E22, SS, RS (PM Only)
1434 + o	Set	Feedback n Velocity Filter BW.	O	-	O	O	O	
1435 + o	Set	Feedback n Acceleration Filter BW	O	-	O	O	O	
256	Set*	Event Checking Control	O	-	O	-	-	
257	Get	Event Checking Status	O	-	O	-	-	
258	Get	Registration 1 Positive Edge Position	R	-	R	-	-	
259	Get	Registration 1 Negative Edge Position	R	-	R	-	-	
260	Get	Registration 2 Positive Edge Position	O	-	O	-	-	
261	Get	Registration 2 Negative Edge Position	O	-	O	-	-	
262	Get	Registration 1 Positive Edge Time	O	-	O	-	-	
263	Get	Registration 1 Negative Edge Time	R	-	R	-	-	
264	Get	Registration 2 Positive Edge Time	R	-	R	-	-	
265	Get	Registration 2 Negative Edge Time	O	-	O	-	-	
266	Get	Home Event Position	R	-	R	-	-	
268	Get	Home Event Time	O	-	O	-	-	
93	Get	Command Target Time	-	-	O	-	-	
286	Set*	Controller Position Command	-	-	R	-	-	
287	Set*	Controller Velocity Command	-	-	O	R	-	
288	Set*	Controller Acceleration Command	-	-	O	O	O	
289	Set*	Controller Torque Command	-	-	O	O	O	
290	Get	Interpolated Command Position	-	-	O	-	-	
291	Get	Interpolated Command Velocity	-	-	O	O	-	
292	Get	Interpolated Command Accel.	-	-	O	O	O	
91	Set*	Command Data Set	-	-	R	R	R	
92	Set*	Interpolation Control	-	-	R	R	-	
302	Set	Velocity Limit - Positive	-	-	O	R	-	
303	Set	Velocity Limit - Negative	-	-	O	R	-	
304	Set	Acceleration Limit	-	-	O	R	-	
305	Set	Deceleration Limit	-	-	O	R	-	
306	Set	Jerk Limit Control	-	-	O	O	-	
307	Set	Flying Start Enable	-	-	O	O	-	
308	Set	Skip Speed 1	-	O	-	-	-	
309	Set	Skip Speed 2	-	O	-	-	-	
310	Set	Skip Speed 3	-	O	-	-	-	
311	Set	Skip Speed Band	-	O	-	-	-	
420	Get	Position Command	-	-	R	-	-	

**Motion Axis Object, Class Code: 42<sub>hex</sub>**

Instance Attribute			Implementation by Drive Category Code					
Attr. ID	Access Rule	Attribute Name	N	F	C - PI Vector Ctrl			Conditional
			V	P	V	T	Implementation	
421	Set*	Position Trim	-	-	R	-	-	
422	Get	Position Reference	-	-	R	-	-	
423	Get	Velocity Feedforward Command	-	-	R	-	-	
424	Get	Position Feedback	R	-	R	R	O	
426	Get	Position Error	-	-	R	-	-	
427	Get	Position Integrator Output	-	-	R	-	-	
428	Get	Position Loop Output	-	-	R	-	-	
430	Set	Kvff	-	-	R	-	-	
431	Set	Kpp	-	-	R	-	-	
432	Set	Kpi	-	-	R	-	-	
433	Set	Position Lock Tolerance	-	-	R	-	-	
434	Set	Position Error Tolerance	-	-	R	-	-	
435	Set	Position Error Tolerance Time	-	-	O	-	-	
439	Set	Position Integrator Control	-	-	R	-	-	
440	Set	Position Integrator Preload	-	-	R	-	-	
450	Get	Velocity Command	-	R	R	R	-	
451	Set*	Velocity Trim	-	R	R	R	-	
452	Get	Accel. Feedforward Command	-	-	R	R	-	
453	Get	Velocity Reference	-	R	R	R	-	
454	Get	Velocity Feedback	R	-	R	R	O	
455	Get	Velocity Error	-	-	R	R	-	
456	Get	Velocity Integrator Output	-	-	R	R	-	
457	Get	Velocity Loop Output	-	-	R	R	-	
460	Set	Kaff	-	-	R	-	-	
461	Set	Kvp	-	-	R	R	-	
462	Set	Kvi	-	-	R	R	-	
464	Set	Kdr	-	R	O	R	-	
465	Set	Velocity Error Tolerance	-	-	O	O	-	
466	Set	Velocity Error Tolerance Time	-	-	O	O	-	
467	Set	Velocity Integrator Control	-	-	R	R	-	
468	Set	Velocity Integrator Preload	-	-	R	R	-	
469	Set	Velocity Low Pass Filter Bandwidth	-	-	O	O	-	
470	Set	Velocity Threshold	-	O	R	R	-	
471	Set	Velocity Lock Tolerance	-	O	R	R	-	
472	Set	Velocity Standstill Window	-	R	R	R	-	
480	Get	Acceleration Command	-	-	O	O	O	
481	Set*	Acceleration Trim	-	-	O	O	O	
482	Get	Acceleration Reference	-	-	O	O	O	
483	Get	Acceleration Feedback	R	-	R	R	O	
490	Get	Torque Command	-	-	R	R	R	
491	Set*	Torque Trim	-	-	R	R	R	
492	Get	Torque Reference	-	-	R	R	R	

Motion Axis Object, Class Code: 42<sub>hex</sub>

Instance Attribute			Implementation by Drive Category Code					
Attr. ID	Access Rule	Attribute Name	N	F	C - PI Vector Ctrl			Conditional
			V	P	V	T	Implementation	
493	Get	Filtered Torque Reference	-	-	R	R	R	
494	Get	Limited Torque Reference	-	-	R	R	R	
496	Set	K <sub>j</sub>	-	-	R	R	O	
498	Set	Friction Compensation	-	-	O	O	O	
502	Set	Torque LP Filter Bandwidth	-	-	O	O	O	
503	Set	Torque Cmd Notch Frequency	-	-	O	O	O	
504	Set	Torque Limit - Positive	-	-	R	R	R	
505	Set	Torque Limit - Negative	-	-	R	R	R	
506	Set	Torque Rate Limit	-	-	O	O	O	
507	Set	Torque Threshold	-	-	O	O	O	
508	Set	Overtorque Limit	-	-	O	O	O	
509	Set	Overtorque Limit Time	-	-	O	O	O	
510	Set	Undertorque Limit	-	-	O	O	O	
511	Set	Undertorque Limit Time	-	-	O	O	O	
520	Get	I <sub>q</sub> Current Command	-	-	R	R	R	
521	Get	Operative Current Limit	-	-	O	O	O	
522	Get	Current Limit Source	-	-	O	O	O	
523	Get	Motor Electrical Angle	-	-	R	R	R	
524	Get	I <sub>q</sub> Current Reference	-	-	O	O	O	
525	Get	I <sub>d</sub> Current Reference	-	-	O	O	O	
527	Get	I <sub>q</sub> Current Error	-	-	O	O	O	
528	Get	I <sub>d</sub> Current Error	-	-	O	O	O	
529	Get	I <sub>q</sub> Current Feedback	-	-	O	O	O	
530	Get	I <sub>d</sub> Current Feedback	-	-	O	O	O	
531	Get	I <sub>q</sub> Decoupling	-	-	O	O	O	
532	Get	I <sub>d</sub> Decoupling	-	-	O	O	O	
533	Get	V <sub>q</sub> Voltage Output	-	-	O	O	O	
534	Get	V <sub>d</sub> Voltage Output	-	-	O	O	O	
535	Get	U Voltage Output	-	-	O	O	O	
536	Get	V Voltage Output	-	-	O	O	O	
537	Get	W Voltage Output	-	-	O	O	O	
538	Get	U Current Feedback	-	-	O	O	O	
539	Get	V Current Feedback	-	-	O	O	O	
540	Get	W Current Feedback	-	-	O	O	O	
541	Get	U Current Offset	-	-	O	O	O	
542	Get	V Current Offset	-	-	O	O	O	
543	Get	W Current Offset	-	-	O	O	O	
554	Get	K <sub>qp</sub>	-	-	O	O	O	
555	Get	K <sub>qi</sub>	-	-	O	O	O	
556	Get	K <sub>dp</sub>	-	-	O	O	O	
557	Get	K <sub>di</sub>	-	-	O	O	O	
558	Set	Flux Up Control	-	-	O	O	O	Induction Motor only

Motion Axis Object, Class Code: 42<sub>hex</sub>

Instance Attribute			Implementation by Drive Category Code					
Attr. ID	Access Rule	Attribute Name	N	F	C - PI Vector Ctrl			Conditional
			V	P	V	T	Implementation	
559	Set	Flux Up Time	-	-	O	O	O	Induction Motor only
570	Get	Slip Compensation	-	R	-	-	-	
575	Set	Frequency Control Method	-	R	-	-	-	
571	Set	Rated Slip Speed	-	R	-	-	-	
572	Set	Maximum Voltage	-	R	-	-	-	
573	Set	Maximum Frequency	-	R	-	-	-	
574	Set	Frequency Limit	-	R	-	-	-	
575	Set	Break Voltage	-	R	-	-	-	
576	Set	Break Frequency	-	R	-	-	-	
577	Set	Start Boost	-	R	-	-	-	
578	Set	Run Boost	-	R	-	-	-	
600	Get	Output Frequency	-	R	-	-	-	
601	Get	Output Current	-	R	R	R	R	
602	Get	Output Voltage	-	R	R	R	R	
603	Get	Output Power	-	R	R	R	R	
604	Set	PWM Frequency	-	O	O	O	O	
610	Set	Stopping Mode	-	R	R	R	R	
611	Set	Stopping Current	-	-	R	R	R	
612	Set	Stopping Time Limit	-	-	R	R	R	
613	Set	Resistive Brake Contact Delay	-	O	O	O	O	PM Motors Only
614	Set	Mechanical Brake Control	-	O	O	O	O	
615	Set	Mechanical Brake Release Delay	-	R	R	R	R	
616	Set	Mechanical Brake Engage Delay	-	R	R	R	R	
620	Get	DC Bus Voltage	-	R	R	R	R	
621	Get	DC Bus Voltage - Nominal	-	R	R	R	R	
624	Set	Bus Regulator Action	-	R	R	R	R	
625	Set	Bus Regulator Power Limit	-	O	O	O	O	
627	Set	Power Loss Action	-	O	O	O	O	
628	Set	Power Loss Threshold	-	O	O	O	O	
629	Set	Shutdown Action	-	O	O	O	O	
635	Get	Motor Capacity	-	R	R	R	R	
636	Get	Inverter Capacity	-	R	R	R	R	
637	Get	Converter Capacity	-	O	O	O	O	
638	Get	Bus Regulator Capacity	-	O	O	O	O	
639	Get	Ambient Temperature	-	O	O	O	O	
640	Get	Inverter Heatsink Temperature	-	O	O	O	O	
641	Get	Inverter Temperature	-	O	O	O	O	
642	Get	Motor Temperature	-	O	O	O	O	
643	Get	Feedback 1 Temperature	-	O	O	O	O	
644	Get	Feedback 2 Temperature	-	O	O	O	O	
645	Get	Inverter Overload Limit	-	O	O	O	O	
646	Set	Motor Overload Action	-	O	O	O	O	

Motion Axis Object, Class Code: 42<sub>hex</sub>

Instance Attribute			Implementation by Drive Category Code				
Attr. ID	Access Rule	Attribute Name	N	F	C - PI Vector Ctrl		Conditional
			V	P	V	T	Implementation
647	Set	Inverter Overload Action	-	O	O	O	O
650	Get	Axis State	R	R	R	R	R
651	Get	Axis Status	R	R	R	R	R
652	Get	Axis Status - Mfg	R	R	R	R	R
653	Get	Axis I/O Status	R	R	R	R	R
654	Get	Axis I/O Status - Mfg	R	R	R	R	R
94	Set*	Status Data Set	R	R	R	R	R
655	Get	Axis Exception Status	R	R	R	R	R
656	Get	Axis Exception Status - Mfg	R	R	R	R	R
657	Get	Axis Fault Status	R	R	R	R	R
658	Get	Axis Fault Status - Mfg	R	R	R	R	R
659	Get	Axis Alarm Status	R	R	R	R	R
660	Get	Axis Alarm Status - Mfg	R	R	R	R	R
661	Get	Axis Fault Code	R	R	R	R	R
662	Get	Axis Fault Code - Mfg	R	R	R	R	R
663	Get	Axis Fault Time Stamp	R	R	R	R	R
664	Get	Axis Alarm Code	R	R	R	R	R
665	Get	Axis Alarm Code - Mfg	R	R	R	R	R
666	Get	Fault Sub Code	O	O	O	O	O
680	Get	Motor Overtemperature Factory Limit	-	O	O	O	O
681	Get	Motor Thermal Overload Factory Limit	-	O	O	O	O
682	Get	Inverter Overtemperature Factory Limit	-	O	O	O	O
683	Get	Inverter Thermal Overload Factory Limit	-	O	O	O	O
684	Get	Converter Overtemperature Factory Limit	-	O	O	O	O
685	Get	Converter Thermal Overload Factory Limit	-	O	O	O	O
686	Get	Bus Regulator Overtemperature Factory Limit	-	O	O	O	O
687	Get	Bus Regulator Temperature Overload Factory Limit	-	O	O	O	O
688	Get	Bus Overvoltage Factory Limit	-	O	O	O	O
689	Get	Bus Undervoltage Factory Limit	-	O	O	O	O
695	Set	Motor Overspeed User Limit	-	O	O	O	O
696	Set	Motor Overtemperature User Limit	-	O	O	O	O
697	Set	Motor Temperature Overload User Limit	-	O	O	O	O
698	Set	Inverter Overtemperature User Limit	-	O	O	O	O
699	Set	Inverter Temperature Overload User Limit	-	O	O	O	O
700	Set	Converter Overtemperature User Limit	-	O	O	O	O

Motion Axis Object, Class Code: 42<sub>hex</sub>

Instance Attribute			Implementation by Drive Category Code					
Attr. ID	Access Rule	Attribute Name	N	F	C - PI Vector Ctrl			Conditional
			V	P	V	T	Implementation	
701	Set	Converter Thermal Overload User Limit	-	O	O	O	O	
702	Set	Bus Regulator Overtemperature User Limit	-	O	O	O	O	
703	Set	Bus Regulator Temperature Overload User Limit	-	O	O	O	O	
704	Get	Bus Overvoltage User Limit	-	O	O	O	O	
705	Get	Bus Undervoltage User Limit	-	O	O	O	O	
667	Set	Exception Action	R	R	R	R	R	
668	Set	Exception Action - Mfg	R	R	R	R	R	
669	Get	Initialization Fault Status	R	R	R	R	R	
670	Get	Initialization Fault Status - Mfg	R	R	R	R	R	
671	Get	Initialization Fault Code	R	R	R	R	R	
672	Get	Initialization Fault Code - Mfg	R	R	R	R	R	
673	Get	Start Inhibit Status	-	R	R	R	R	
674	Get	Start Inhibit Status - Mfg	-	R	R	R	R	
675	Get	Start Inhibit Code	-	R	R	R	R	
676	Get	Mfg Start Inhibit Code	-	R	R	R	R	
710	Get	Control Power-up Time	-	O	O	O	O	
711	Get	Cumulative Run Time	-	O	O	O	O	
712	Get	Cumulative Energy Usage	-	O	O	O	O	
713	Get	Cumulative Motor Revs	-	O	O	O	O	
714	Get	Cumulative Main Power Cycles	-	O	O	O	O	
715	Get	Cumulative Control Power Cycles	-	O	O	O	O	
720	Get	Inverter Rated Output Voltage	-	R	R	R	R	
721	Get	Inverter Rated Output Current	-	R	R	R	R	
722	Get	Inverter Rated Output Power	-	R	R	R	R	

Attribute ID Offset, o = (n-1)\*50

**5-46.5 Class Attributes**

The following table of attributes applies to the Motion Axis Object class, which may be referenced as instance 0. These attributes exist even before any Motion Axis Object instances have been created. Since they are not tied to any particular axis instance of a CIP Motion Device, the class attributes are generally used to address parametric behavior that applies to all axis instances, e.g. the communications node behavior. As instances of this class are created, they are given consecutive instance numbers starting at 1.

Notes:

1. No attributes associated with this object require Non-Volatile storage, so a “No” is implied for all attributes under the NV column.
2. Vendor specific bits, and enumerations provide space for device vendors to provide additional product features.

**Motion Axis Object, Class Code: 42<sub>hex</sub>****Key to Table:** (See Sections 5-46.3 and 5-46.4 for details.)

(R) – Required bit or enumeration – must be supported in the implementation

(O) – Optional bit or enumeration – support is left to vendor's discretion

Set\* - Indicates the attribute is normally set by the CIP Motion connection data block and not by a Set service.

**Table 5-46.2 Class Attributes for the Motion Axis Object**

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
1 thru 7	These class attributes are optional and are described in Chapter 4 of the Volume 1 of the CIP Networks Library.						
10	Required	Set*		Controller Consumed Connection Data	Struct	This attribute contains the consumed data structure defined by the CIP Motion Controller-to-Device Connection.	See Semantics.
11	Required	Set*		Controller Produced Connection Data	Struct	This attribute contains the produced data structure defined by the CIP Motion Device-to-Controller Connection.	See Semantics.
14	Required	Set*		Node Control	BYTE	Contains bits used to control the behavior of the associated device communications node.	See Semantics.
15	Required	Get		Node Status	BYTE	Contains bits used to indicate the status of the associated device communications node:	See Semantics.
16	Required	Get		Node Fault	USINT	Contains numeric code of active node fault condition. The Node Fault Code is used to provide diagnostic detail to pin-point the source of the fault condition.	See Semantics.
17	Optional	Get		Node Alarm	USINT	Contains numeric code of the current active alarm condition. The Node Alarm Code is used to provide diagnostic detail to pin-point the source of the alarm condition.	See Semantics.
18	Required	Set*		Controller Update Period	UDINT	Represents the period between updates of the controller that is also the period between CIP Motion Controller-to-Device Connection updates.	Nanoseconds
19	Required	Set*		Controller Time Offset	LINT	This element represents the 64-bit value at the beginning of the Controller Update Period that is associated with the Controller Time Stamp value. The Controller Time Offset is the value that is added to the controller's local clock to produce System Time.	Nanoseconds

Motion Axis Object, Class Code: 42<sub>hex</sub>

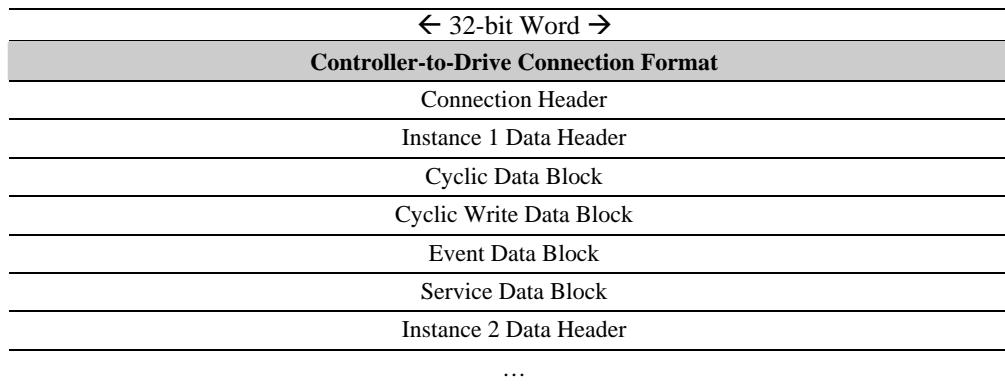
Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
20	Required	Set*		Controller Time Stamp	LINT	This element represents the 64-bit System Time value at the beginning of the Controller Update Period when the controller's update timer event occurred. It is calculated by the controller as the sum of the controller's local clock value when update timer event occurred and the controller's System Time Offset value given by Controller Time Offset. The Controller Time Stamp is therefore directly associated with the command data contained in the connection. The time stamp format is absolute and follows the CIP Sync standard with 0 corresponding to January 1, 1970.	Nanoseconds (CIP Sync absolute)
21	Optional	Set		Controller Update Delay High Limit	USINT	Represents high limit delay threshold for a Controller-to-Device Connection update. This delay is specified in units of Controller Update Periods. Exceeding this limit results in a Control Update Fault.	# of Controller Update Periods
22	Optional	Set		Controller Update Delay Low Limit	USINT	Represents high limit delay threshold allowed for a Controller-to-Device Connection update. This delay is specified in units of Controller Update Periods. Exceeding this limit results in a Control Update Alarm.	# of Controller Update Periods
28	Optional	Get		Sync Variance	UDINT	Represents the current statistical variation of the System Time Offset value from the mean System Time Offset value.	Nanoseconds
29	Optional	Set		Sync Threshold	UDINT	Determines the threshold for the Observed Variance of System Time below which the Motion Axis Object is considered synchronized. The Group Sync service uses this as a criterion for a successful response.	Nanoseconds Default: device dependent minimum value.

Motion Axis Object, Class Code: 42<sub>hex</sub>

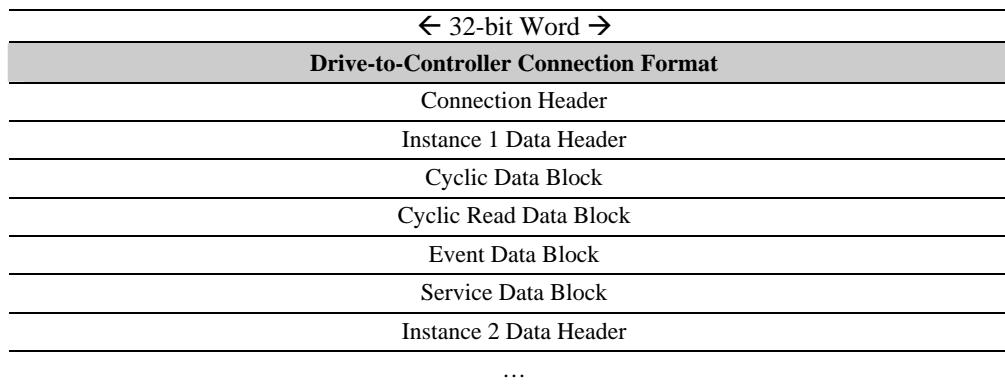
Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
30	Optional	Get		Sync Update Delay	REAL	This value is the product of the number of networks hops the device is away from the grandmaster, as determined by the Time Sync object attribute, Steps Removed, and the time synchronization interval, as determined by the Time Sync object attribute, Sync Interval.	Seconds
31	Optional	Set		Time Data Set	BYTE	This bit-mapped byte contains flags that determine the usage and format of the controller and device timing information. This value is derived from Time Configuration element sent by the controller in the Controller-to-Device Connection and becomes the Time Configuration value sent by the device in the Device-to-Controller Connection.	Bit Field: 0 = Update Period 1 = Time Stamps 2 = Time Diagnostics 3-7 = Reserved

**5-46.6 Semantics****5-46.6.1 Attribute #10 - Controller Consumed Connection Data**

This attribute contains the consumed data structure defined by the CIP Motion Controller-to-Device Connection. The content of this dynamic data structure is updated by the controller at the Controller Update Period. This structure, outlined in the figure below, is dynamically defined by connection header data. Refer to the Connections section for a detailed description of the connection data structure.

**Figure 5-46.8 Controller Consumed Connection Data Format****5-46.6.2 Attribute #11 - Controller Produced Connection Data**

This attribute contains the produced data structure defined by the CIP Motion Device-to-Controller Connection. This dynamic data structure is updated by the CIP Motion Device and transmitted to the controller at the Controller Update Period. This structure, outlined in the figure below, is dynamically defined by connection header data. Refer to the Connections section for a detailed description of the connection data structure.

**Figure 5-46.9 Controller Produced Connection Data Format**

### 5-46.6.3 Attribute #14 – Node Control

Contains bits used to control the behavior of the associated motion device communications node.

**Table 5-46.3 Node Control Bit Definitions**

Bit	Req. Opt.	Name	Description
0	R	Remote Control	This bit is used to request that the CIP Motion Device turn over complete control to the controller. If this bit is clear, the CIP Motion Device is under the exclusive control of the local interface; the CIP Motion controller cannot affect the behavior of the device in any way other than to switch the device back to Remote Control mode. If the CIP Motion connection is lost while the device is under remote control, the device may then be controlled via the local interface.
1	O	Sync Control	This bit is used by the controller to request synchronous operation of the CIP Motion Device. Synchronous operation is defined as having the device node's local timer synchronized with System Time and that the device node start using the Controller Time Stamp and Time Offset to process the connection data and schedule future Device-to-Controller Connection updates. The Synchronous Control bit implies that the Controller Time Stamp and Time Offset values are valid. The bit shall remain set for the duration of synchronous operation. In asynchronous mode, there is no requirement for the local timer to be synchronized nor is time-stamping necessary. The associated time data is not valid
2	R	Controller Data Valid	This bit must be set for the CIP Motion Device to process the instance data blocks. During the connection initialization sequence there may be a period of time where the connection data in the instance data blocks is not yet initialized. This condition can be recognized by the device by first checking this bit and finding it clear.
3	R	Node Fault Reset	This bit is used to request that the CIP Motion Device perform a reset of the communications node and attempt to clear the Node Fault bit. When the controller detects that the Node Fault bit is clear, the Node Fault Reset bit is cleared. If the communication fault persists, the Node Fault bit may stay set, in which case a power cycle or a complete reconfiguration process must be performed
4-5			Reserved
6-7			Vendor Specific

#### 5-46.6.4 Attribute #15 – Node Status

Contains bits used to indicate the status of the associated drive communications node:

**Table 5-46.4 Node Status Bit Definitions**

Bit	Required Optional	Name	Description
0	R	Remote Mode	The Remote Mode bit is used to indicate that the CIP Motion Device has turned over total control to the controller. If this bit is clear, the device must not act on any data contained in the Controller-to-Device cyclic or service channels.
1	O	Sync Mode	The Sync Mode bit indicates whether or not the CIP Motion Device is synchronized. Synchronous operation is defined as having the CIP Motion Device's local timer synchronized with System Time and that the CIP Motion Device is using the connection time stamps to process the connection data. The Sync Mode bit being set also implies that the Device Time Stamp is valid. The bit shall remain set for the duration of synchronous operation. If the Sync Mode bit is clear the CIP Motion Device is said to be in Asynchronous mode. In Asynchronous mode, there is no requirement for the local timer to be synchronized nor is time-stamping necessary or even valid.
2	R	Data Valid	The Data Valid bit must be set for the controller to process the instance data blocks from the CIP Motion Device. During the connection initialization sequence there may be a period of time where the connection data in the instance data blocks is not yet initialized. This condition can be recognized by the controller by first checking this bit.
3	R	Node Fault	The Node Fault bit is used to indicate that the CIP Motion Device has detected one or more fault conditions related to the communications node. Specific fault conditions can be determined by the Node Fault attribute. If this bit is clear, there are no fault conditions present. The Node Fault bit may be cleared by setting the Node Fault Reset bit in the Node Control word. All associated axes are disabled when a Node Fault condition is present.
4-5			Reserved
6-7			Vendor Specific

#### 5-46.6.5 Attribute #16 – Node Faults

This attribute is an 8-bit fault code used to indicate the presence of specific fault condition associated with the device's communications node.

**Table 5-46.5 Node Fault Code Definitions**

Code	Name	Description
0	No Fault	Indicates that there is no fault condition currently present at the device communications node.
1	Control Update Fault	The Control Update Fault bit is used to indicate that updates from the controller have been excessively late.
2	Processor Watchdog Fault	The Processor Watchdog Fault indicates that the processor associated with this CIP Motion Device has experienced an excessive overload condition that has tripped the associated processor watchdog mechanism.
3	Hardware Fault	The Hardware Fault indicates that the critical support hardware (FPGA, ASIC, etc.) associated with the device node has experienced a fault condition.

**Motion Axis Object, Class Code: 42<sub>hex</sub>**

<b>Code</b>	<b>Name</b>	<b>Description</b>
4	Data Format Error	This fault indicates that an error has occurred in the data format between the controller and the device. Eg: a format revision mismatch would be one example which would set this fault.
5-127	Reserved	
128-255	Vendor Specific	

**5-46.6.6 Attribute #17 – Node Alarms**

This attribute is an 8-bit alarm code used to indicate specific alarm conditions of the associated device's communications node. Alarms do not result in any direct action.

**Table 5-46.6 Node Alarm Code Definitions**

<b>Code</b>	<b>Name</b>	<b>Description</b>
0	No Alarm	Indicates that there is no alarm condition currently present at the device communications node.
1	Control Update Alarm	The Control Update Alarm bit is used to indicate that updates from the controller have been late.
2	Processor Overload Alarm	The Processor Overload Alarm indicates that the processor associated with device is experiencing overload conditions that could eventually lead to a fault
3	Sync Alarm	Sync Alarm indicates that the Sync Variance has exceeded the Sync Threshold while the device is running in Sync Mode
4-127	Reserved	
128-255	Vendor Specific	

**5-46.7 Instance Attributes**

This section lists all the supported attributes of a Motion Axis Object instance. Because of the large number of attributes listed in this section, an attempt has been made to organize the attributes by functional category. Each functional grouping may be further organized by first listing the object Status and Signal attributes, followed by the object Configuration attributes.

Notes:

1. Due to the large number of instance attributes supported by this object, 16-bit Attribute IDs are required.
2. No attributes associated with this object require Non-Volatile storage, so a "No" is implied for all attributes under the NV column.
3. Unless otherwise specified, all optional attributes default to 0.

**Key to Tables:** (See Sections 5-46.3 and 5-46.4 for details.)

(R) – Required bit or enumeration – must be supported in the implementation

(O) – Optional bit or enumeration – support is left to vendor's discretion

**Control Modes:**

P – Position Control

V – Velocity Control

T – Torque Control

N – No Control

**Control Methods:**

**F** – Frequency Drives (V/Hz or VFD)

**C** – Closed Loop (Vector Control)

Set\* - Indicates the attribute is normally set by the CIP Motion connection header and not by a Set service.

% Device Rated Units – defined as the percentage of the continuous rating of the device with 100% implying operation at the continuous rated specification for the device. This unit can be applied to attributes related to speed, torque, force, current, voltage, and power. Applicable “Devices” can be Motor, Inverter, Converter, or Bus Regulator. Note that this unit can be used independent of whether the attribute value represents an instantaneous level or a time-averaged level; the appropriate unit for the device rating is implied. As with all attributes that are in units of %, an attribute value of 100 means 100%.

Feedback Units: Attributes that relate to motion dynamics typically express displacement in terms of the selected feedback device count resolution. Since the device can use different feedback channels for different control loops depending on the feedback configuration, the determination of which feedback device applies depends on the Feedback Configuration. A cross-reference table is provided below to determine the appropriate feedback counts or units to apply to an attribute based on its Dynamic Unit type (position, velocity, or acceleration), and the configured Feedback Configuration. When an axis instance is configured for No Feedback, i.e. sensorless operation (V/Hz, sensorless velocity servo, or future stepper support), feedback counts do not apply, so motion dynamics must be expressed in terms of motor displacement.

**Table 5-46.7 Dynamic Unit vs. Feedback Configuration**

<b>Dynamic Units</b>	<b>Feedback Configuration</b>			
	<b>No Feedback</b>	<b>Feedback 1</b>	<b>Feedback 2</b>	<b>Dual Loop Feedback</b>
<b>Position Control Units</b>	Motor Counts	Feedback 1 Counts	Feedback 2 Counts	Feedback 2 Counts
<b>Velocity Control Units</b>	Motor Units	Feedback 1 Units	Feedback 2 Units	Feedback 1 Units
<b>Accel Control Units</b>	Motor Units	Feedback 1 Units	Feedback 2 Units	Feedback 1 Units

## 5-46.8 Motion Control Configuration Attributes

The following attribute table contains basic motion control configuration attributes associated with a Motion Axis Object instance. These attributes govern aspects of the overall behavior of the motion axis object.

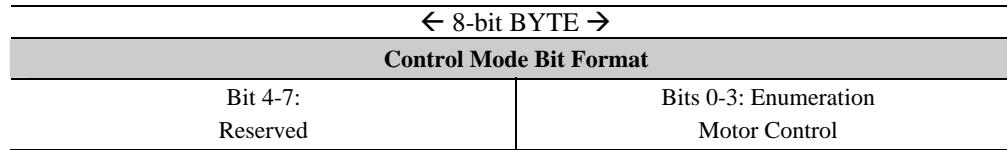
Set\*- These attributes are generally updated via the cyclic Command Data Set of the Controller-to-Device Connection. When included as cyclic command data, these attributes should not be updated via a Set service.

**Motion Axis Object, Class Code: 42<sub>hex</sub>****Table 5-46.8 Motion Control Configuration Attributes**

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
80	Required - All	Set*		Control Mode	BYTE	Attribute that determines the general dynamic control behavior of the motion axis instance.	See Semantics
81	Required - All	Set		Control Method	USINT	Enumerated code that determines the basic motor control algorithm applied by the device to control the dynamic behavior of the motor.	See Semantics

**5-46.8.1 Semantics****5-46.8.2 Attribute #1 – Control Mode**

The Control Mode attribute consists of a 4-bit enumerated Motor Control field and 4-bits reserved for future expansion.

**Figure 5-46.10 Control Mode Bit Field**

The Motor Control field enumeration determines the specific dynamic behavior of the motor that the device is to control for this axis instance. The system view of these control modes are described in detail in Chapter 5-46.2.3. Here is a brief summary of the Motor Control modes:

**Table 5-46.9 Motor Control Field Enumeration Definitions**

Enum.	Req. Opt.	Name	Description
0	R/N	No Control	No motor control is provided in this mode but interface to a specific feedback device as a master feedback source is possible via the Feedback Configuration attribute.
1	R/P	Position Control	Drive seeks to control the position, or orientation, of the motor.
2	R/PV	Velocity Control	Drive seeks to control the velocity of the motor.
3	O	Acceleration Control	Drive seeks to control the acceleration of the motor.
4	R/C	Torque Control	Drive seeks to control the torque output of the motor.
5	O/C	Current Control	Drive seeks to control the torque producing current to the motor.
6-15		Reserved	

**5-46.8.3 Attribute #2 – Control Method**

The Control Method attribute is an 8-bit enumerated code that determines the basic control algorithm applied by the device to control the dynamic behavior of the motor associated with an axis instance.

Table 5-46.10 Control Method Field Enumeration Definitions

Enum.	Req. Opt.	Name	Description
0	R/N	No Control	No Control is associated with a Control Mode of No Control where there is no explicit motor control provided by the device for this axis instance.
1	R/F	Frequency Control	Frequency Control is an “open loop” control method that applies voltage to the motor, generally in proportion to the commanded frequency or speed. This control method is associated with Variable Frequency Drives (VFDs) or so called Volts/Hertz drives.
2	R/C	PI Vector Control	PI Vector Control is a “closed loop” control method that uses actual or estimated feedback for closed loop cascaded PI control of motor dynamics, i.e. position, velocity, acceleration, and torque, and always includes independent closed loop PI control of I <sub>q</sub> and I <sub>d</sub> components of the motor current vector.
3-255		Reserved	-

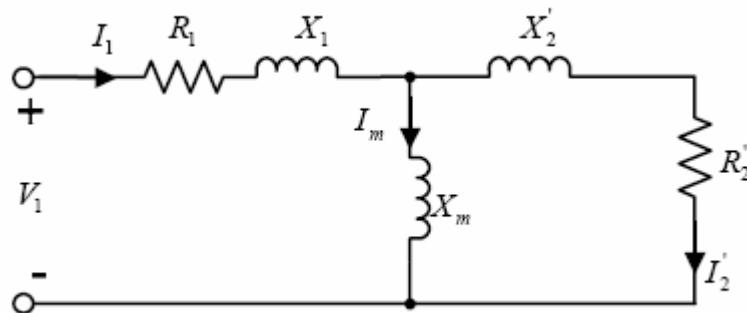
## 5-46.9 Motor Attributes

The following attribute tables contain motor configuration attributes associated with a Motion Axis Object instance that apply to various motor technologies. These motor technologies include three-phase motor rotary, linear, permanent magnet and induction motors. Motor attributes are, therefore, organized according to the various motor types. The Need in Implementation category for an attribute is based on the context of the Motor Type and, thus, to the context of the table in which the attribute appears. Where needed, Standard vs. Optional can be further differentiated by abbreviations for PM (Permanent Magnet) and IM (Induction Motors). It is implied that these motor attributes are not required for No Control (Feedback Only) control mode.

The goal of this motor attribute section is to define the minimal set of required attributes to support device interchangeability. This guarantees that there is sufficient parametric data provided by the controller for any drive compliant with the CIP Motion Device Profile, to effectively control a given motor.

For induction motors, the Motion Axis Object leverages the IEEE recommended phase-neutral equivalent circuit motor model based on “Wye” configuration. Reactance values, X, are related to their corresponding Inductance values, L, by  $X = \omega L$ , where  $\omega$  is the rated frequency of the motor. The prime notation, e.g. X<sub>2'</sub>, R<sub>2'</sub>, indicates that the actual rotor component values X<sub>2</sub> and R<sub>2</sub> are referenced to the stator side of the stator-to-rotor winding ratio.

Figure 5-46.11 IEEE per Phase Motor Model



**Motion Axis Object, Class Code: 42<sub>hex</sub>**

For permanent magnet motors, the Motion Axis Object assumes all motor parameters are defined in the context of a phase-to-phase motor model.

**5-46.9.1 General Motor Attributes**

The following attribute tables contain general motor attributes that apply to all motor technologies.

**Table 5-46.11 General Motor Info Attributes**

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
1310	Optional	Get		Motor Catalog Number	SHORT STRING	A 32-character string that specifies the motor catalog number. This attribute is not applicable when the Motor Data Source is Controller NV.	e.g.
1311	Optional	Get		Motor Serial Number	SHORT STRING	A 16-character string that specifies the serial number of the motor. This attribute is applicable only when the Motor Data Source is Motor NV.	e.g. 0012003400560078
1312	Optional	Get		Motor Date Code	SHORT STRING	A 16-character string that specifies the manufacturing date of the motor. This attribute is applicable only when the Motor Data Source is Motor NV.	e.g.

**Table 5-46.12 General Motor Configuration Attributes**

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
1313	Required	Set		Motor Data Source	USINT	An enumeration that specifies the source of motor data for the drive. Controller NV implies that the motor attributes are derived from the controller's non-volatile memory during the drive configuration process. Drive NV implies that the motor attributes are derived directly from the drive's non-volatile memory. In this mode, no motor parameters shall be sent by the controller to the drive. Motor NV implies that the motor attributes are derived from non-volatile memory of a motor-mounted smart feedback drive equipped with a serial interface. Again, in this mode, no motor parameters shall be sent by the controller to the drive.	0 = Controller NV (R) 1 = Drive NV (O) 2 = Motor NV (O) 3-127 = reserved 128-255 = vendor specific
1314	Required	Set		Motor Device Code	UDINT	A unique number assigned to a motor catalog number. This value is used to insure that the motor and integral motor mounted feedback drive configuration data delivered	

Motion Axis Object, Class Code: 42<sub>hex</sub>

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
						from the controller matches the actual motor and feedback data connected to the drive. This comparison is only valid in the case where the Motor Data Source is Controller NV but the motor is equipped with a smart feedback drive. If the codes do not match, a negative acknowledge is given by the drive. Motor Device Codes are assigned by the motor manufacturer.	
1315	Required	Set		Motor Type	USINT	An enumeration that specifies the motor type.	Enumeration: 0 = rotary permanent magnet (O) 1 = rotary induction (O) 2 = linear permanent magnet (O) 3 = linear induction (O) 4-127 = reserved 128-255 = vendor specific
1316	Required	Set		Motor Unit	UINT	Unit of measure for motor displacement. This is also used for sensorless operation.	Enumeration: 0 = Rev 1 = Meter 2-127 = reserved 128-255 = vendor specific
1317	Optional	Set		Motor Polarity	USINT	An enumerated value used to establish the direction of motor motion in response to positive drive output. Normal polarity is defined as the direction of motor travel when the UVW motor leads are hooked up according to the drive's published specifications. Reverse polarity effectively switches the UVW phasing to UWV so that the motor moves in the opposite direction in response to a positive drive output. This attribute can be used to make the direction of travel agree with the user's definition of positive travel and can be used in conjunction with the Feedback Polarity bit to provide negative feedback, when closed loop control is required.	Enumeration: 0 = Normal Polarity 1 = Reverse Polarity
1318	Required	Set		Motor Rated Voltage	REAL	A float that specifies the nameplate AC voltage rating of the motor. This represents the phase-to-phase voltage applied to the motor to reach rated speed at full load.	Volts (RMS)

Motion Axis Object, Class Code: 42<sub>hex</sub>

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
1319	Required	Set		Motor Rated Continuous Current	REAL	A float that specifies the nameplate AC continuous current rating of the motor. This represents the current applied to the motor under full load conditions at rated speed and voltage.	Amps (RMS)
1320	Required – PM Optional - IM	Set		Motor Rated Peak Current	REAL	A float that specifies the peak or intermittent current rating of the motor. The peak current rating of the motor is often determined by either the thermal constraints of the stator winding or the saturation limits of PM motor magnetic material.	Amps (RMS)
1321	Required – IM Optional – PM	Set		Motor Rated Output Power	REAL	A float that specifies the nameplate rated output power rating of the motor. This represents the power output of motor under full load conditions at rated current, speed and voltage.	Watts
1322	Required	Set		Motor Overload Limit	REAL	A float that specifies the maximum thermal overload limit for the motor. This value is typically 100%, corresponding to the power dissipated when operating at the continuous current rating of the motor, but can be significantly higher if cooling options are applied. For induction motors, this attribute is related to the Service Factor of the motor. When the Motor Capacity attribute value associated with the motor thermal model exceeds the Motor Overload Limit, the drive can optionally trigger a predetermined Motor Overload action. The Motor Overload Limit can also be used by the drive to determine the absolute thermal capacity limit of the motor, i.e. the Motor Thermal Overload Factory Limit, that if exceeded, generates a Motor Thermal Overload FL exception.	% Motor Rated
1323	Optional	Set		Motor Integral Thermal Switch	BOOL	A boolean that specifies if the motor has an integral thermal switch. 0 = No 1 = Yes	0 = No 1 = Yes
1324	Optional	Set		Motor Max Winding Temperature	REAL	A float that specifies the maximum winding temperature of the motor.	°C
1325	Optional	Set		Motor Winding-to-Ambient Thermal Capacitance	REAL	A float that specifies the winding-to-ambient thermal capacitance.	Joules/°C

**Motion Axis Object, Class Code: 42<sub>hex</sub>**

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
1326	Optional	Set		Motor Winding-to-Ambient Thermal Resistance	REAL	A float that specifies the winding-to-ambient thermal resistance.	°C/Watt

**5-46.9.2 General PM Motor Attributes**

The following attribute table contains motor configuration attributes that apply to Permanent Magnet motor types in general.

**Table 5-46.13 General PM Motor Configuration Attributes**

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
1327	Required	Set		PM Motor Resistance	REAL	A float that specifies the phase-to-phase, resistance of a permanent magnet motor.	Ohms
1328	Required	Set		PM Motor Inductance	REAL	A float that specifies the phase-to-phase, inductance of a permanent magnet motor.	Henries

**5-46.9.3 General Rotary Motor Attributes**

The following attribute table contains motor configuration attributes that apply specifically to rotary motor types.

**Table 5-46.14 General Rotary Motor Configuration Attributes**

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
1329	Required	Set		Rotary Motor Poles	UINT	An integer that specifies the number of poles per revolution for rotary motors. This value is always an even number, as poles always exist in pairs.	
1330	Optional	Set		Rotary Motor Inertia	REAL	A float that specifies the unloaded inertia of a rotary motor.	Kg·m <sup>2</sup>
1331	Required	Set		Rotary Motor Rated Speed	REAL	A float that specifies the nameplate rated speed of a rotary motor at rated voltage and rated current. For induction motors this value is often referred to as base speed. The value also generally represents the speed at which the output power of the motor is maximized.	RPM
1332	Optional	Set		Rotary Motor Max Speed	REAL	A float that specifies the absolute maximum speed of a rotary motor in units of RPM. This speed may be determined by the limitations of the motor or by limitations of the mechanical system. Specifically, this value can represent the maximum safe operating speed, maximum continuous “no-load” speed, maximum continuous encoder speed, or maximum continuous bearing speed of the motor. This value can be used to determine the Motor Overspeed exception.	RPM

**Motion Axis Object, Class Code: 42<sub>hex</sub>**

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
1333	Optional	Set		Rotary Motor Damping Coefficient	REAL	A float that specifies the damping, or viscous friction, associated with a rotary motor.	N-meters/(radian/second)

**5-46.9.4 General Linear Motor Attributes**

The following attribute table contains motor configuration attributes that apply specifically to linear motor types.

**Table 5-46.15 General Linear Motor Configuration Attributes**

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
1334	Required	Set		Linear Motor Pole Pitch	REAL	A float that specifies the pole pitch of a linear motor in units of meters, and is equivalent to the electrical cycle length.	Meters
1335	Required	Set		Linear Motor Rated Speed	REAL	A float that specifies the nameplate rated speed of a linear motor at rated voltage and rated current. For induction motors this value is often referred to as base speed. The value also generally represents the speed at which the rated output power of the motor is measured..	Meters/sec
1336	Optional	Set		Linear Motor Moving Mass	REAL	A float that specifies the unloaded moving mass of a linear motor.	Kg
1337	Optional	Set		Linear Motor Max Speed	REAL	A float that specifies the absolute maximum speed of a linear motor in units of m/s. This speed may be determined by the limitations of the motor or by limitations of the mechanical system. Specifically, this value can represent the maximum safe operating speed, maximum continuous “no-load” speed, maximum continuous encoder speed, or maximum continuous bearing speed of the motor. This value can be used to determine the Motor Overspeed exception.	Meters/sec
1338	Optional	Set		Linear Motor Damping Coefficient	REAL	A float that specifies the damping, or viscous friction, associated with a linear motor.	N/(m/s)

### 5-46.9.5 Rotary PM Motor Attributes

The following attribute table contains motor configuration attributes that apply specifically to rotary motor types.

**Table 5-46.16 Rotary PM Motor Configuration Attributes**

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
1339	Required	Set		PM Motor Rated Torque	REAL	A float that specifies the nameplate continuous torque rating of a rotary permanent magnet motor.	N-m
1340	Required	Set		PM Motor Torque Constant	REAL	A float that specifies the torque constant of a rotary permanent magnet motor in Newton-meters per RMS Amp.	N-m/Amp (RMS)
1341	Required	Set		PM Motor Rotary Voltage Constant	REAL	A float that specifies the voltage, or back-EMF, constant of a rotary permanent magnet motor in phase-to-phase RMS Volts per KRPM.	Volts (RMS) / KRPM

### 5-46.9.6 Linear PM Motor Attributes

The following attribute table contains motor configuration attributes that apply specifically to linear PM motor types.

**Table 5-46.17 Linear PM Motor Configuration Attributes**

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
1342	Required	Set		Motor Rated Force	REAL	A float that specifies the nameplate continuous force rating of a linear permanent magnet motor.	N
1343	Required	Set		Motor Force Constant	REAL	A float that specifies the force constant of a linear permanent magnet motor in Newtons per RMS Amp.	N/Amp (RMS)
1344	Required	Set		Motor Linear Voltage Constant	REAL	A float that specifies the voltage, or back-EMF, constant of a linear permanent magnet motor in phase-to-phase RMS Volts per meter/sec.	Volts (RMS) / (m/s)

### 5-46.9.7 Induction Motor Attributes

The following attribute table contains motor configuration attributes that apply specifically to induction motor types.

Motion Axis Object, Class Code: 42<sub>hex</sub>

Table 5-46.18 Induction Motor Configuration Attributes

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
1345	Required	Set		Induction Motor Rated Frequency	REAL	A float that specifies the nameplate frequency rating of an induction motor.	Hertz
1346	Required	Set		Induction Motor Flux Current	REAL	Id Current Reference that required to generate full motor flux. This value is closely approximated by the No Load Motor Rated Current commonly found in Induction Motor data sheets.	Amps (RMS)
1347	Required	Set		Induction Motor Stator Resistance	REAL	A float that specifies the Y circuit, phase-neutral, winding resistance of the stator as shown as R1 in the IEEE motor model.	Ohms
1348	Required	Set		Induction Motor Stator Leakage Reactance	REAL	A float that specifies the Y circuit, phase-neutral, leakage reactance of the stator winding, at rated frequency, as shown as X1 in the IEEE motor model.	Ohms
1349	Required*	Set		Induction Motor Magnetization Reactance	REAL	A float that specifies the Y circuit, phase-neutral, magnetizing reactance of the motor, at rated frequency, as shown as Xm in the IEEE motor model.	Ohms
1350	Required*	Set		Induction Motor Rotor Resistance	REAL	A float that specifies the phase-neutral equivalent stator-referenced winding resistance of the rotor as shown as R2' in the IEEE motor model.	Ohms
1351	Required	Set		Induction Motor Rotor Leakage Reactance	REAL	A float that specifies the Y circuit, phase-neutral, equivalent stator-referenced leakage inductance of the rotor winding, at rated frequency, as shown as X2' in the IEEE motor model.	Ohms

\* These parameters have a strong motor temperature component that some drives circumvent through various adaption or compensation techniques. Nevertheless these parameters are classified as required for the purposes of drive interoperability.

## 5-46.10 Feedback Attributes

The following attribute tables contain all position feedback related attributes associated with a Motion Axis Object instance that apply to various feedback device and feedback interface technologies. These feedback interface technologies include Digital AqB (digital A quad B signals), Sine/Cosine (analog A quad B signals), Digital Parallel (parallel digital bit interface), SSI (Synchronous Serial Interface), LDT (Linear Displacement Transducer) and Resolver. Other modern feedback interfaces supported are Hiperface® (by Stegmann) and EnDat 2.1® & EnDat 2.2® (by Heidenhain). The “Need in Implementation” category for an attribute is based on the context of the Feedback Type abbreviations for TT (Digital AqB), TP (Digital Parallel), SC (Sine/Cosine), HI (Hiperface®), E21 (EnDat 2.1®), E22 (EnDat 2.2®), SS (SSI), LT (LDT – Linear Displacement Transducer, and RS (Resolver). The abbreviation HS applies to all devices that support Hall Sensors.

**Motion Axis Object, Class Code: 42<sub>hex</sub>**

The goal of this feedback attribute section is to define the minimal set of required attributes to support device interchangeability. This guarantees that there is sufficient parametric data provided by the controller for any drive compliant with the CIP Motion Drive Profile to effectively interface to a wide range of feedback device types.

Multiple feedback device interfaces are currently defined by the Motion Axis Object for control loop feedback, namely Feedback 1 and Feedback 2. In dual feedback control configurations, Feedback 1 is generally associated with the motor mounted feedback device while Feedback 2 is associated with the load-side or machine mounted feedback device. For all other control modes, Feedback 1 must be used for the singular feedback interface and is always required for PM Motor commutation.

When No Control is selected for a Motion Axis Object instance, any uncommitted feedback channel can be used as a master feedback source. Since Feedback 1 is typically assigned to the control function of the primary Motion Axis Object instance, Feedback 2, 3, 4, etc., are likely candidates as master feedback sources. For this version of the Motion Axis Object, two additional feedback channels, Feedback 3 and Feedback 4, are supported for this purpose.

To minimize the length of the feedback attribute tables below, the letter “n” in the generic “Feedback n” attribute name is used to specify the associated feedback channel number. Valid channel numbers for open standard feedback attributes of the Motion Axis Object are 1, 2, 3, and 4. Attribute IDs are assigned based on the channel number. Support for feedback interface channels, 2, 3, 4 is optional in the device implementation. However, if hardware support for any of these feedback channels is available in a given device, these optional attributes are clearly required in the implementation for the channels to be available for use.

**Table 5-46.19 General Feedback Info Attributes**

GENERAL FEEDBACK INFO ATTRIBUTES							
Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
1400 + (n-1)*50	Optional - All	Get		Feedback n Catalog Number	SHORT STRING	A 32-character string that specifies the catalog number of the device associated with Feedback n.	e.g.
1401 + (n-1)*50	Optional - All	Get		Feedback n Serial Number	SHORT STRING	A 16-character string that specifies the serial number of the device associated with Feedback n.	e.g. 0012003400560078
1402 + (n-1)*50	Required - NPVC Optional - T	Get		Feedback n Position	DINT	Actual position of the axis based on Feedback n.	Feedback n Counts
1403 + (n-1)*50	Required - NPVC Optional - T	Get		Feedback n Velocity	REAL	Actual filtered velocity of the axis based on Feedback n.	Feedback n Units / Sec
1404 + (n-1)*50	Required -NPVC Optional - T	Get		Feedback n Acceleration	REAL	Actual filtered acceleration of the axis based on Feedback n.	Feedback n Units / Sec <sup>2</sup>

Motion Axis Object, Class Code: 42<sub>hex</sub>

Table 5-46.20 Feedback Configuration Attributes

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
90	Required – PVT	Set*		Feedback Data Set	BYTE	Bit mapped field that determines what actual feedback data values are being transmitted to the controller in the Drive-to-Controller connection.	Bit Field: 0 = Position Feedback (R/C) 1 = Velocity Feedback (R/C) 2 = Accel Feedback (O) 3 = Torque Reference (R/T) 4 = Iq Current Reference (O) 5 = Voltage Output (O) 6 = Frequency Output (O) 7 = Reserved
82	Required - All	Set*		Feedback Configuration	BYTE	This attribute determines how the various available feedback channels are used to implement the selected Control Mode.	See Semantics
83	Required - All	Set*		Feedback Master Select	USINT	This attribute determines what Logical channel is assigned to this axis instance when the Feedback Configuration is set to Master Feedback.	Enumeration: 0 = Reserved 1 = Feedback 1 2 = Feedback 2 3 = Feedback 3 4 = Feedback 4
84	Optional - PC	Set		Feedback 1/2 Count Ratio	REAL	Number of Feedback 1 Counts per Feedback 2 Counts. This value is used to convert between feedback 2 counts to feedback 1 counts when configured for dual position loop operation.	Feedback 1 Counts per Feedback 2 Count
1411 + (n-1)*50	Optional - NC	Set		Feedback n Unit	USINT	Unit of measure for the designated feedback device. The Feedback Unit for Feedback 1 must be scalable to the configured Motor Unit; if the Motor Unit is set to Rev, Feedback 1 Unit must be set to Rev; if Motor Unit is set to Meter, Feedback 1 Unit must be set to either Meter or Inch and the drive must handle conversion between the linear motor and feedback displacement units.	Enumeration: 0 = Rev 1 = Meter 2 = Inch 3-255 = reserved

**Motion Axis Object, Class Code: 42<sub>hex</sub>**

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
1412 + (n-1)*50	Optional	Set		Feedback n Port Select	REAL	Maps the logical Feedback Channel “n” to a physical Feedback Port “m”. Default Feedback n Port Select = n.	Enumeration: 0 = Reserved 1 = Port 1 2 = Port 2 3 = Port 3 4 = Port 4 5-255 = Reserved
1413 + (n-1)*50	Required	Set		Feedback n Type	USINT	Identifies the type of feedback device connected to the associated Feedback interface. Drive support for any individual feedback types is left to the discretion of the drive manufacturer. However if a specific feedback type is supported, attributes associated with that type are generally required in the implementation.	Enumeration: 0 = No Feedback Present 1 = Digital AqB (w/o Hall) 2 = Digital AqB (w/ Hall) 3 = Digital Parallel (Absolute) 4 = Sine/Cosine (w/o Hall) 5 = Sine/Cosine (w/ Hall) 6 = Hiperface ® 7 = EnDat 2.1 ® 8 = EnDat 2.2 ® 9 = Resolver 10 = SSI 11 = LDT 12-127 = reserved 128-255 = vendor specific

**Motion Axis Object, Class Code: 42<sub>hex</sub>**

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
1414 + (n-1)*50	Optional	Set		Feedback n Polarity	USINT	An enumerated value used to establish the direction of change in the feedback counter in response to positive motion of the associated feedback device. Normal polarity is defined as that which results in increasing feedback counts when the feedback device is hooked up and moved in the positive direction according to the drives published specifications. Reverse polarity internally switches the polarity of the feedback accumulator so that the feedback counts decrease when the feedback device moves in the positive direction. This attribute can be used to make the direction of travel agree with the user's definition of positive travel and can be used in conjunction with the Motor Polarity bit to provide negative feedback, when this feedback channel is used for closed loop control.	Enumeration: 0 = Normal Polarity 1 = Reverse Polarity
1415 + (n-1)*50	Required	Set		Feedback n Mode	USINT	Determine how the drive applies the feedback count value.  When configured for Incremental mode, the drive zeroes the feedback count accumulator at power-up.  When configured for Absolute mode, the drive initializes the feedback count accumulator at power-up to the absolute feedback position value read from the feedback device.	Enumeration: 0 = Incremental 1 = Absolute

Motion Axis Object, Class Code: 42<sub>hex</sub>

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
1416 + (n-1)*50	Required - TT, SC, HI, E21, RS	Set		Feedback n Cycle Resolution	UDINT	Number of feedback cycles per Feedback Unit. Cycles for a Digital AqB device is the “line” resolution of the encoder, while it represents the sinusoidal “cycle” resolution of a Sin/Cos feedback device and the “pole” count of Resolver feedback device. For digital serial (e.g. SSI) or parallel absolute feedback device interfaces, this value would represent the “step” resolution of the device.	Feedback Cycles / Feedback Unit
1417 + (n-1)*50	Required - TT, SC, HI, E21, RS	Set		Feedback n Cycle Interpolation	UDINT	Number of interpolated Feedback Counts per Feedback Cycle. For a Digital AqB device the drive’s feedback interface hardware can generally support interpolation values of 1, 2, or 4. For a Sin/Cos, Hiperface, Endat 2.1, or Resolver feedback device the number is generally much larger and determined by the interpolation capability of the drive feedback interface hardware. A value of 1024 is typical in this case. For digital serial (e.g. SSI) or parallel absolute feedback device interfaces, this value is always 1 since there is no opportunity of drive-based interpolation. The effective resolution of the feedback device in Feedback Counts per Feedback Unit is the product of the Feedback Cycle Resolution and the Feedback Cycle Interpolation.	Feedback Counts / Feedback Cycle
1420 + (n-1)*50	Required – TP, SS, HI, E21, E22	Set		Feedback n Data Length	USINT	Number of feedback data bits transferred over the digital serial or parallel data interface channel of a feedback device.	# of Bits

**Motion Axis Object, Class Code: 42<sub>hex</sub>**

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
1421 + (n-1)*50	Optional – TP, SS	Set		Feedback n Data Code	USINT	Type of feedback data bit encoding used by designated serial or parallel data interface channel of a feedback device.	Enumeration: 0 = Binary 1 = Gray
1422 + (n-1)*50	Optional - RS	Set		Feedback n Resolver Transformer Ratio	REAL	Transformer Ratio specification of the designated resolver feedback device.	
1423 + (n-1)*50	Optional - RS	Set		Feedback n Resolver Excitation Voltage	REAL	Sets the sinusoidal excitation voltage applied to the rotor of the designated resolver feedback device.	Volts (RMS)
1424 + (n-1)*50	Optional - RS	Set		Feedback n Resolver Excitation Frequency	REAL	Frequency of sinusoidal excitation signal applied to the designated resolver feedback device. Valid frequency range or values for this attribute depends on the specific drive hardware interface.	Hertz
1425 + (n-1)*50	Optional - RS	Set		Feedback n Resolver Cable Balance	REAL	Adjusts the relative amplitude of the Sine and Cosine signals from the resolver to compensate for impact of resolver cable.	%
1426 + (n-1)*50	Required - LT	Set		Feedback n LDT Type	USINT	Determines the LDT type. Options are Start/Stop and PWM. Start/Stop transducers accept an input (interrogate) signal to start the measurement cycle and respond with two pulses on the Return line. Timing can be based on either the Rising or Falling edge. The time between the pulses is proportional to the position. PWM transducers respond to the interrogate signal with a single long pulse on the Return line. The pulse width is proportional to the position.	Enumeration: 0 = PWM 1 = Start/Stop Rising 2 = Start/Stop Falling
1427 + (n-1)*50	Required - LT	Set		Feedback n LDT Recirculations	USINT	Determines the number of recirculations associated with a PWM type LDT transducer. Multiple recirculations can be used to increase the resolution of the LDT at the expense of increasing the sample period.	# Recirculations

Motion Axis Object, Class Code: 42<sub>hex</sub>

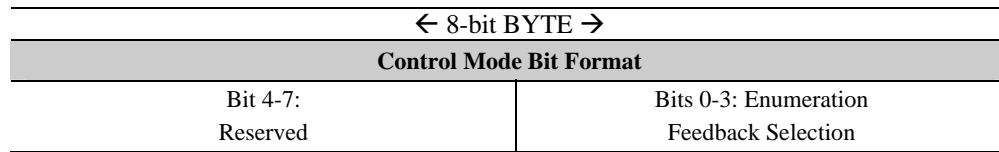
Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
1428 + (n-1)*50	Required – HS, TP, HI, E21, E22, SS, RS (PM Motors Only)	Set		Feedback n Commutation Offset	REAL	A value that specifies the commutation offset of the PM motor mounted feedback device in units of electrical degrees. This attribute specifies the offset from a commutation reference position defined by applying DC current into the U terminal and out of the shorted V and W terminals of the motor and allowing the rotor to move to its magnetic null position relative to the stator. On an absolute encoder or resolver, the offset is the difference from the device's zero absolute position and the commutation reference position. On an incremental encoder with Hall sensors, the offset is the difference between the position corresponding to a transition of the Hall S3 channel (with the S1 channel high and the S2 channel low) and the commutation reference position. The commutation offset is only applicable to the motor mounted Feedback 1 device.	Electrical Degrees
1434 + (n-1)*50	Optional	Set		Feedback n Velocity Filter Bandwidth.	REAL	Controls the bandwidth of the Low Pass Filter applied to the raw velocity signal from Feedback n. A value of 0 for this attribute disabled this feature.	Radians/sec
1435 + (n-1)*50	Optional	Set		Feedback n Accel Filter Bandwidth	REAL	Controls the bandwidth of the Low Pass Filter applied to the raw acceleration signal from Feedback n. A value of 0 for this attribute disabled this feature.	Radians/sec

### 5-46.10.1 Semantics

### 5-46.10.2 Attribute #3 – Feedback Configuration

This attribute contains a 4-bit enumerated Feedback Selection field that determines how the various logical feedback channels are used to implement the selected Control Mode for this axis instance.

**Figure 5-46.12 Feedback Configuration Bit Field**



Feedback Selection enumerations provide support for multi-feedback control functionality for the various active Control Modes, i.e. where the device is actively controlling the motor based on feedback. In these active Control Modes it is assumed that logical channel, Feedback 1, is attached directly to the motor while Feedback 2 is attached to the load side of the mechanical transmission. Commutation signals for a PM motor are always derived from the Feedback 1.

**Table 5-46.21 Feedback Selection Field Enumeration Definitions**

Enum.	Req. Opt.	Name	Description
0	R/NF O/C	No Feedback	No Feedback is selected when sensorless open loop or closed loop control is desired. When performing open loop control, no feedback signal is required. In closed loop control, the required feedback signal is estimated by a sensorless control algorithm based on motor phase voltage and current signals..
1	R/N	Master Feedback	Master Feedback assigns an uncommitted feedback channel, as specified by the Feedback Master Select attribute, to this motion axis instance to serve as a “master feedback” source when the drive is configured for No Control mode
2	R/C	Motor Feedback	When Motor Feedback is selected, then commutation, acceleration, velocity, and position feedback signals are all derived from motor mounted Feedback 1
3	O/C	Load Feedback	When Load Feedback is selected, then motor-mounted Feedback 1 is only used for PM motor commutation while load-side Feedback 2 is used for position, velocity, and acceleration.
4	O/P	Dual Feedback	When Dual Feedback is selected, then motor mounted Feedback 1 is used for commutation, acceleration, and velocity, and load-side Feedback 2 is used strictly for position.
5-255		Reserved	

### 5-46.11 Event Capture Attributes

The following attribute tables contain all event related attributes associated with a Motion Axis Object instance. These include registration, marker, and homing events. The Event Capture attributes are designed to support the possibility of up to 16 active events per controller update period. The format of all Time Stamp attributes is absolute System Time and follows the CIP Sync standard with 0 corresponding to January 1, 1970.

**Motion Axis Object, Class Code: 42<sub>hex</sub>**

The Motion Axis Object currently supports two independent registration input channels per axis instance that can be triggered on either the rising or falling edges of the signal. If the device hardware implementation allows, event time and position data can be captured for all four event conditions simultaneously. The Event Capture attributes also support Auto-rearm for registration events. This allows for controller implementation of important features like Windowed Registration and Registration Pattern Recognition.

The Motion Axis Object also supports Home Switch, Marker and Switch-Marker events for homing functionality on a per axis basis. The Marker events are typically generated by the configured position feedback device for the associated axis instance.

**Set\*** - These attributes are generally updated via the cyclic Command Data Set of the Controller-to-Device Connection. When included as cyclic command data, these attributes should not be updated via a Set service.

**Table 5-46.22 Event Attributes**

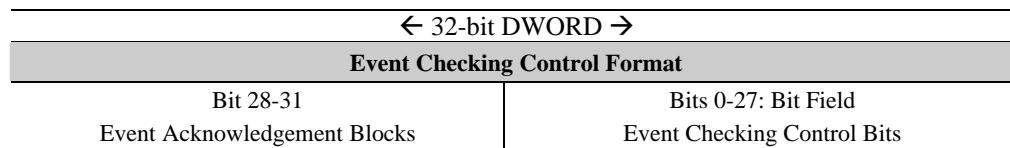
Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
256	Optional – NP	Set*		Event Checking Control	DWO RD	This attribute is passed to the drive by the controller as part of the Drive to Controller connection for the purpose of arming various drive inputs, e.g. marker, home switch, and registration inputs, to generate events to the controller. When these enabled events occur, the drive captures both the time and exact axis position when the event occurred.	See Semantics
257	Optional – NP	Get		Event Checking Status	DWO RD	This attribute is passed to the drive by the controller as part of the Drive to Controller connection to indicate if the drive is currently checking for events based on various drive inputs, e.g. marker, home, and registration inputs. Event checking is initiated when the corresponding Event Checking Control bit is set in the controller to drive connection.	See Semantics
258	Required – NP	Get		Registration 1 Positive Edge Position	DINT	Feedback position latched on the rising edge of the Registration Input 1.	Position Control Units
259	Required – NP	Get		Registration 1 Negative Edge Position	DINT	Feedback Position latched on the falling edge of the Registration Input 1.	Position Control Units
260	Optional – NP	Get		Registration 2 Positive Edge Position	DINT	Feedback position latched on the rising edge of the Registration Input 2.	Position Control Units
261	Optional – NP	Get		Registration 2 Negative Edge Position	DINT	Feedback Position latched on the falling edge of the Registration Input 2.	Position Control Units

**Motion Axis Object, Class Code: 42<sub>hex</sub>**

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
262	Optional – NP	Get		Registration 1 Positive Edge Time	LINT	System Time stamp on the rising edge of the Registration Input 1.	Nanoseconds (CIP Sync absolute)
263	Required – NP	Get		Registration 1 Negative Edge Time	LINT	System Time stamp on the falling edge of the Registration Input 1.	Nanoseconds (CIP Sync absolute)
264	Required – NP	Get		Registration 2 Positive Edge Time	LINT	System Time stamp on the rising edge of the Registration Input 2.	Nanoseconds (CIP Sync absolute)
265	Optional – NP	Get		Registration 2 Negative Edge Time	LINT	System Time stamp on the falling edge of the Registration Input 2.	Nanoseconds (CIP Sync absolute)
266	Required – NP	Get		Home Event Position	DINT	Feedback Position latched on the specified home event.	Position Control Units
268	Optional – NP	Get		Home Event Time	LINT	System Time stamp latched on the specified home event.	Nanoseconds (CIP Sync absolute)

**5-46.11.1 Semantics****5-46.11.2 Attribute #256 – Event Checking Control**

The first 28-bits of this 32-bit attribute are used to enable various device inputs, e.g. marker, home switch, and registration inputs, to generate events to the controller. When these enabled events occur, the device captures both the time and exact position of the associated motion axis instance. The most significant 4-bits represent the number of Event Acknowledgement messages are in the Event Data Block in the last Controller-to-Device Connection update. For a detailed description of the event handling protocol between the drive and the controller refer to Connection section in the appropriate CIP Motion device profile.

**Figure 5-46.13 Event Checking Control Word Field**

For the Home Switch-Marker events shown in the figure below, the device first looks for level of home switch input to transition according to the first + or – symbol and then immediately look for the transition of the marker input according to the second + or – symbol.

**Table 5-46.23 Event Checking Control Bit Definitions**

Bit	Req. Opt.	Name	Description
0	R/P	Reg 1 Pos Edge	Enable checking for positive edge transition of Registration 1 Input
1	R/P	Reg 1 Neg Edge	Enable checking for negative edge transition of Registration 1 Input
2	O/P	Reg 2 Pos Edge	Enable checking for positive edge transition of Registration 2 Input
3	O/P	Reg 2 Neg Edge	Enable checking for negative edge transition of Registration 2 Input
4-7		Reserved	

Motion Axis Object, Class Code: 42<sub>hex</sub>

Bit	Req. Opt.	Name	Description
8	O/P	Reg 1 Pos Auto-rearm	Enable auto-rearm mechanism after positive edge transition of Registration 1 Input
9	O/P	Reg 1 Neg Auto-rearm	Enable auto-rearm mechanism after negative edge transition of Registration 1 Input
10	O/P	Reg 2 Pos Auto-rearm	Enable auto-rearm mechanism after positive edge transition of Registration 2 Input
11	O/P	Reg 2 Neg Auto-rearm	Enable auto-rearm mechanism after negative edge transition of Registration 2 Input
12-15		Reserved	
16	R/P	Marker Pos Edge	Enable checking for positive edge of Marker input of associated position feedback channel.
17	R/P	Marker Neg Edge	Enable checking for negative edge of Marker input of associated position feedback channel.
18	R/P	Home Switch Pos Edge	Enable checking for positive edge of Home input associated with this axis instance.
19	R/P	Home Switch Neg Edge	Enable checking for positive edge of Home input associated with this axis instance.
20	R/P	Home Switch-Marker ++	Enable checking for event sequence specified as a positive edge transition of the Home Switch input followed by a positive edge of the Marker input.
21	R/P	Home Switch-Marker +-	Enable checking for event sequence specified as a positive edge transition of the Home Switch input followed by a negative edge of the Marker input.
22	R/P	Home Switch-Marker - +	Enable checking for event sequence specified as a negative edge transition of the Home Switch input followed by a positive edge of the Marker input.
23	R/P	Home Switch-Marker - -	Enable checking for event sequence specified as a negative edge transition of the Home Switch input followed by a negative edge of the Marker input.
24-27		Reserved	

**5-46.11.3 Attribute #257 – Event Checking Status**

The first 28-bits of this 32-bit attribute are used to indicate if the device is currently checking for events based on various device inputs, e.g. marker and registration inputs. Event checking is initiated when the corresponding event checking control bit is set in the Controller-to-Device Connection. When an event occurs, the device captures both the time and exact axis position and passes this information to the controller in event notification data blocks. But for the controller to process the event data, the corresponding event checking status bit must be set. The most significant 4-bits of this attribute represent the number of Event Notification messages are in the Event Data Block in the last Device-to-Controller Connection update. For a detailed description of the event handling protocol between the device and the controller refer to Connection section.

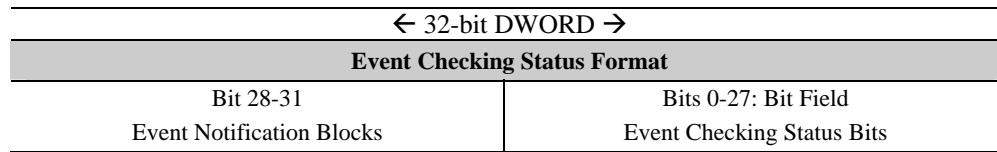
**Figure 5-46.14 Event Checking Status Word Field**

Table 5-46.24 Event Checking Status Bit Definitions

Bit	Req. Opt.	Name	Description
0	R/P	Reg 1 Pos Edge	Checking for positive edge transition of Registration 1 Input?
1	R/P	Reg 1 Neg Edge	Checking for negative edge transition of Registration 1 Input?
2	O/P	Reg 2 Pos Edge	Checking for positive edge transition of Registration 2 Input?
3	O/P	Reg 2 Neg Edge	Checking for negative edge transition of Registration 2 Input?
4-7		Reserved	-
8	O/P	Reg 1 Pos Auto-rearm	Auto-rearm mechanism after positive edge transition of Registration 1 Input active?
9	O/P	Reg 1 Neg Auto-rearm	Auto-rearm mechanism after negative edge transition of Registration 1 Input active?
10	O/P	Reg 2 Pos Auto-rearm	Auto-rearm mechanism after positive edge transition of Registration 2 Input active?
11	O/P	Reg 2 Neg Auto-rearm	Auto-rearm mechanism after negative edge transition of Registration 2 Input active?
12-15		Reserved	
16	R/P	Marker Pos Edge	Checking for positive edge of Marker input of associated position feedback channel?
17	R/P	Marker Neg Edge	Checking for negative edge of Marker input of associated position feedback channel?
18	R/P	Home Switch Pos Edge	Checking for positive edge of Home input associated with this axis instance?
19	R/P	Home Switch Neg Edge	Checking for positive edge of Home input associated with this axis instance?
20	R/P	Home Switch-Marker ++	Checking for event sequence specified as a positive edge transition of the Home Switch input followed by a positive edge of the Marker input?
21	R/P	Home Switch-Marker +-	Checking for event sequence specified as a positive edge transition of the Home Switch input followed by a negative edge of the Marker input?
22	R/P	Home Switch-Marker --	Checking for event sequence specified as a negative edge transition of the Home Switch input followed by a positive edge of the Marker input?
23	R/P	Home Switch-Marker --	Checking for event sequence specified as a negative edge transition of the Home Switch input followed by a negative edge of the Marker input?
24-27		Reserved	

### 5-46.12 Command Reference Generation Attributes

The following lists of attributes apply to the command reference generation functionality of the device that converts command position, velocity, acceleration, and torque data output from a controller-based planner into corresponding command references signals to the device's motor control structures. The command reference generator functionality includes fine interpolators, signal selector switches, dynamic limiters.

Set\* - These attributes are generally updated via the cyclic Command Data Set of the Controller-to-Device Connection. When included as cyclic command data, these attributes should not be updated via a Set service.

Motion Axis Object, Class Code: 42<sub>hex</sub>

Table 5-46.25 Command Generator Signal Attributes

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
93	Required – P Optional - VT	Get		Command Target Time	LINT	Time stamp associated with command data from the motion planner and used by the command fine interpolators. The Command Target Time is the sum of the Controller Time Stamp value and the Command Target Update interval associated with this axis instance. The time stamp format is absolute and follows the CIP Sync standard with 0 corresponding to January 1, 1970.	Nanoseconds (CIP Sync absolute)
286	Required - P	Set*		Controller Position Command	DINT	Command position data value from controller, updated at the Controller Update Period, and feeding into a fine interpolator.	Position Control Units
287	Required – V Optional – P	Set*		Controller Velocity Command	REAL	Command velocity data value from controller, updated at the Controller Update Period, and feeding into a fine interpolator.	Velocity Control Units / Sec
288	Optional - PVT	Set*		Controller Acceleration Command	REAL	Command acceleration data value from controller, updated at the Controller Update Period, and feeding into a fine interpolator.	Accel Control Units / Sec <sup>2</sup>
289	Optional - PVT	Set*		Controller Torque Command	REAL	Commanded torque data value from controller, updated at the Controller Update Period, and feeding into a fine interpolator.	% Motor Rated
290	Optional - P	Get		Interpolated Command Position	DINT	Output value from the Command Position fine interpolator.	Position Control Units
291	Optional - PV	Get		Interpolated Command Velocity	REAL	Output value from the Command Velocity fine interpolator. When no Command Velocity signal is present when performing position control, this signal can be derived by scaling the Differential Position output value of the Command Position fine interpolator.	Velocity Control Units / Sec
292	Optional - PVT	Get		Interpolated Command Acceleration	REAL	Output value from the Command Acceleration fine interpolator. When no Command Acceleration signal is present when performing position or velocity control, this signal can be derived by scaling the Differential Velocity output value of the Command Velocity fine interpolator. If no Command Velocity signal is	Accel Control Units / Sec <sup>2</sup>

Motion Axis Object, Class Code: 42<sub>hex</sub>

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
						present, the Interpolated Command Acceleration signal can be derived by scaling the 2 <sup>nd</sup> Differential Position output value of the Command Position fine interpolator.	

Table 5-46.26 Command Generator Configuration Attributes

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
91	Required – PVT	Set*		Command Data Set	BYTE	Bit mapped field that determines what command data values are being updated by the controller.	Bit Field: 0 = Position Command (R/P) 1 = Velocity Command (R/V) 2 = Accel Command (O) 3 = Torque Command (R/T) 4 = Position Trim (O) 5 = Acceleration Trim (O) 6 = Velocity Trim (O) 7 = Torque Trim (O)
92	Required – PV	Set*		Interpolation Control	BYTE	Controls the interpolation/extrapolation method applied to position, velocity, and acceleration command data from the motion planner based on the associated time stamp.	See Semantics
302	Required – V Optional - P	Set		Velocity Limit - Positive	REAL	This value defines the maximum allowable positive velocity command into the velocity summing junction. If this velocity limit value is exceeded, the device responds by clamping the velocity command to this limit and setting the Velocity Limit status bit.	Velocity Control Units / Sec
303	Required – V Optional - P	Set		Velocity Limit - Negative	REAL	This value defines the maximum allowable negative velocity command into the velocity summing junction. If this velocity limit value is exceeded, the devicee responds by clamping the velocity command to this limit and setting the Velocity Limit status bit.	Velocity Control Units / Sec

**Motion Axis Object, Class Code: 42<sub>hex</sub>**

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
304	Required – V Optional - P	Set		Acceleration Limit	REAL	This value defines the maximum allowable positive rate of change in the magnitude of the velocity command (increasing speed). If this acceleration limit value is exceeded, the device responds by clamping the rate of change of the velocity command to this limit and setting the Acceleration Limit status bit.	Velocity Control Units / Sec <sup>2</sup>
305	Required – V Optional - P	Set		Deceleration Limit	REAL	This value defines the maximum allowable negative rate of change in the magnitude of the velocity command (decreasing speed). If this deceleration limit value is exceeded, the device responds by clamping the rate of change of the velocity command to this limit and setting the Deceleration Limit status bit.	Velocity Control Units / Sec <sup>2</sup>
306	Optional - PV	Set		Jerk Limit Control	REAL	When accel or decel limited, this value sets the percentage of accel or decel time that is applied to the speed ramp as jerk limited S Curve. The S Curve time is added $\frac{1}{2}$ at the beginning and $\frac{1}{2}$ at the end of the ramp. A value of 0 results in no S-Curve, i.e. a linear acceleration ramp. A value of 100% results in a triangular acceleration profile with the peak being the acceleration/deceleration limit. As the Jerk Limit Control value increases the derived Jerk limit value decreases.	%

Motion Axis Object, Class Code: 42<sub>hex</sub>

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
307	Optional – PV	Set		Flying Start Enable	BOOL	This Boolean value is used to enable or disable the Flying Start feature of the device. When Flying Start Enabled is true and the motion axis is enabled, the device determines the current velocity of the motor as part of the Starting State initialization activities. Just prior to transitioning to the Running state, the device presets the output of the Accel Limiter to the current velocity. In this way, the motor seamlessly ramps from its current velocity to the commanded velocity profile generated by the motion planner. When Flying Start Enabled is false, the motor velocity is not determined and a preset of 0 is applied to the Accel Limiter output.	False = Flying Start Disabled True = Flying Start Enabled
308	Optional - F	Set		Skip Speed 1	REAL	Sets the central speed of a skip speed band within which the drive does not operate.	Velocity Control Units / Sec
309	Optional - F	Set		Skip Speed 2	REAL	Sets the central speed of a skip speed band within which the drive does not operate.	Velocity Control Units / Sec
310	Optional - F	Set		Skip Speed 3	REAL	Sets the central speed of a skip speed band within which the drive does not operate.	Velocity Control Units / Sec
311	Optional - F	Set		Skip Speed Band	REAL	Determines the speed window around a skip speed that cannot be commanded. Any command set-point within this window is adjusted by the Skip Speed block to fall at either the upper or lower Skip Speed Band boundary value. The axis can smoothly accelerate or decelerate through the skip speed band based on the acceleration limiter block but may not operate at a set speed within the band. The Skip Speed Band is distributed $\frac{1}{2}$ above and $\frac{1}{2}$ below the skip speed. This Skip Speed Band attribute applies to all skip speeds supported in the device. A value of 0 for this attribute disabled this feature.	Velocity Control Units / Sec

### 5-46.12.1 Semantics

### 5-46.12.2 Attribute #301 – Interpolation Control

The Interpolation Control attribute governs the interpolation/extrapolation method applied to position, velocity, and acceleration command data from the motion planner based on the associated time stamp. The first two bits of this 8-bit attribute represent the Command Target Update that determines the number of Update Periods (UP) added to the command data Time Stamp to determine the absolute System Time that the command data value is targeted for, i.e. the Command Target Time. For more details on interpolation and extrapolation control, please refer to the Section 6.6.2 on Command Fine Interpolation.

**Figure 5-46.15 Interpolation Control Word Field**

← 8-bit BYTE →		
Event Checking Status Format		
Bits 6-7: Vendor Specific	Bits 2-5: Reserved	Bits 0-1: Enumeration Command Target Update

**Table 5-46.27 Command Target Update Enumeration Definition**

Enum.	Req. Opt.	Name	Description
0	R/P	Immediate	A Command Target Update of 0 implies that the command data is to be applied to the drive control structure immediately.
1	R/P	Extrapolate (+1 UP)	A Command Target Update of 1 implies that extrapolation is to be used to apply the command data to the drive control structure based on adding 1 Update Period to the command Time Stamp.
2	R/P	Interpolate (+2 UP)	A Command Target Update of 2 implies that fine interpolation is to be used to apply the command data based on adding 2 Update Periods to the command Time Stamp.
3		Reserved	-

### 5-46.13 Control Mode Attributes

#### 5-46.13.1 Position Loop Attributes

The following attributes tables contain all position control related attributes associated with a Motion Axis Object instance.

Set\* - this attribute is generally updated via the cyclic command data set of the Controller-to-Device connection. When included as cyclic command data this attribute should not be updated via a Set service.

**Table 5-46.28 Position Loop Signal Attributes**

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
420	Required – P	Get		Position Command	DINT	Command position output from the fine interpolator (if active) into position loop when configured for position loop control.	Position Control Units

**Motion Axis Object, Class Code: 42<sub>hex</sub>**

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
421	Required – P	Set*		Position Trim	DINT	Additional position command added to the Position Command to generate the Position Reference signal into the position loop summing junction.	Position Control Units
422	Required – P	Get		Position Reference	DINT	Command position reference signal into the position loop summing junction to be compared with a position feedback signal.	Position Control Units
423	Required – PC	Get		Velocity Feedforward Command	REAL	Velocity feedforward reference into velocity loop summing junction.	Velocity Control Units / Sec
424	Required – NPVC Optional – T	Get		Position Feedback	DINT	Position feedback value channeled into the position control summing junction.	Position Control Units
426	Required – PC	Get		Position Error	REAL	Error between commanded and actual position that is the output of the position loop summing junction.	Position Control Units
427	Required – PC	Get		Position Integrator Output	REAL	Output of position integrator representing the contribution of the position integrator to Position Loop Output.	Velocity Control Units / Sec
428	Required – PC	Get		Position Loop Output	REAL	Output of the position loop forward path representing the total control effort of the position loop.	Velocity Control Units / Sec

**Table 5-46.29 Position Loop Configuration Attributes**

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
430	Required – PC	Set		Kvff	REAL	Velocity Feedforward Gain value that multiplies the Velocity Feedforward Command signal before applying it to the velocity loop summing junction.	%
431	Required – PC	Set		Kpp	REAL	Position Proportional Gain value that multiplies the Position Error signal before applying it to the velocity loop summing junction.	Radians/Sec
432	Required – PC	Set		Kpi	REAL	Position Integral Gain value that multiplies the Position Integrator Error signal before applying it to the velocity loop summing junction.	Radians/Sec <sup>2</sup>
433	Required –	Set		Position Lock	REAL	This value establishes a window around the current	Position Control

Motion Axis Object, Class Code: 42<sub>hex</sub>

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
	PC			Tolerance		command position. When the actual position is within this window the Position Lock status bit is set. When actual position falls outside this window, the Position Lock status bit is cleared.	Units
434	Required – PC	Set		Position Error Tolerance	REAL	Determines the absolute maximum Position Error value that can be tolerated without causing an Excessive Position Error exception.	Position Control Units
435	Optional – PVC	Set		Position Error Tolerance Time	REAL	Determines the maximum amount of time that the Position Error Tolerance can be exceeded without generating an exception.	Sec
439	Required – PC	Set		Position Integrator Control	BYTE	This bit field controls the behavior of the position loop integrator while commanding motion via the controller. When the integrator hold enable bit is set, the integrator is held while motion is being commanded with a non-zero velocity. When clear, the integrator runs without qualification. When the auto-preset bit is set, the integrator preload value is automatically loaded with the current velocity command when there is a control mode change between velocity control and position control. If clear, the integrator is loaded with the configured position integrator preload value.	Bit Field: 0: Integrator Hold Enable 1: Auto-preset
440	Required - PC	Set		Position Integrator Preload	REAL	Value assigned to the position integrator when the position control loop is enabled.	Velocity Control Units / Sec

**5-46.13.2 Velocity Loop Attributes**

The following attributes tables contain all velocity control related attributes associated with a Motion Axis Object instance.

Set\* - this attribute is generally updated via the cyclic command data set of the Controller-to-Drive connection. When included as cyclic command data this attribute should not be updated via a Set service.

Motion Axis Object, Class Code: 42<sub>hex</sub>

Table 5-46.30 Velocity Loop Signal Attributes

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
450	Required – PV	Get		Velocity Command	REAL	Command velocity output from fine interpolator (if active) into velocity loop when configured for velocity loop control.	Velocity Control Units / Sec
451	Required – PV	Set*		Velocity Trim	REAL	Additional velocity command added to the velocity loop summing junction.	Velocity Control Units / Sec
452	Required – PVC	Get		Acceleration Feedforward Command	REAL	Acceleration feedforward reference into acceleration loop summing junction.	Velocity Control Units / Sec2
453	Required – PV	Get		Velocity Reference	REAL	Command velocity reference into velocity loop summing junction.	Velocity Control Units / Sec
454	Required – NPVC Optional – T	Get		Velocity Feedback	REAL	Actual velocity of the axis applied to the velocity summing junction based on Control Mode selection.	Velocity Control Units / Sec
455	Required – PVC	Get		Velocity Error	REAL	Error between the velocity reference and velocity feedback value that is the output of the velocity loop summing junction.	Velocity Control Units / Sec
456	Required – PVC	Get		Velocity Integrator Output	REAL	Output of velocity integrator representing the contribution of the velocity integrator to Velocity Loop Output.	Accel Control Units / Sec <sup>2</sup>
457	Required – PVC	Get		Velocity Loop Output	REAL	Output of velocity forward path representing the total control effort of the velocity loop.	Accel Control Units / Sec <sup>2</sup>

Table 5-46.31 Velocity Loop Configuration Attributes

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
460	Required – PC	Set		Kaff	REAL	Acceleration Feedforward Gain value that multiplies the Acceleration Feedforward Command signal before applying it to the velocity loop summing junction.	%
461	Required – PVC	Set		Kvp	REAL	Velocity Proportional Gain value that multiplies the Velocity Error signal before applying it to the acceleration summing junction.	Radians/Sec

Motion Axis Object, Class Code: 42<sub>hex</sub>

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
462	Required – PVC	Set		Kvi	REAL	Velocity Integral Gain value that multiplies the Velocity Integrator Error signal before applying it to the acceleration summing junction.	Radians /Sec <sup>2</sup>
464	Required – V Optional – P	Set		Kdr	REAL	Velocity Droop value that provides compliance to the velocity integrator by subtracting a portion of the velocity loop effort from the velocity error input to the velocity integrator. The presence of the Torque/Force scaling gain, Kj, in the droop signal path allows Velocity Droop to be specified in velocity units per % rated torque output. This parameter is also valid for V/Hz drives and its behavior is nearly identical, but instead of % rated being related to torque, % rated is related to current.	Velocity Control Units / % Rated
465	Optional – PVC	Set		Velocity Error Tolerance	REAL	Determines the absolute maximum Velocity Error value that can be tolerated without causing a Excessive Velocity Error exception.	Velocity Control Units/ Sec
466	Optional – PVC	Set		Velocity Error Tolerance Time	REAL	Determines the maximum amount of time that the Velocity Error Tolerance can be exceeded without generating an exception.	Sec
467	Required – PVC	Set		Velocity Integrator Control	BYTE	This bit field controls the behavior of the velocity loop integrator while commanding motion via the controller. When the integrator hold enable bit is set, the integrator is held while motion is being commanded with a non-zero velocity. When clear, the integrator runs without qualification. When the auto-preset bit is set, the integrator preload value is automatically loaded with the current torque command when there is a control mode change between torque control and velocity control. If clear, the integrator is loaded with the configured velocity integrator preload value.	Bit Field: 0: Integrator Hold Enable 1: Auto-preset

Motion Axis Object, Class Code: 42<sub>hex</sub>

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
468	Required – PVC	Set		Velocity Integrator Preload	REAL	Value assigned to the velocity integrator when the velocity control loop is enabled.	Accel Control Units / Sec <sup>2</sup>
469	Optional – PVC	Set		Velocity Low Pass Filter Bandwidth.	REAL	Controls the bandwidth of the Low Pass Filter applied to the Velocity Error signal. Recommended implementation is a two pole IIR filter. A value of 0 for this attribute disabled this feature.	Radians/sec
470	Optional – PV	Set		Velocity Threshold	REAL	Defines a minimum absolute velocity, below which, results in the Velocity Threshold status bit being set.	Velocity Control Units / Sec
471	Optional – PV	Set		Velocity Lock Tolerance	REAL	This value establishes a window around the current command velocity. When the actual velocity is within this window the Velocity Lock status bit is set. When actual position falls outside this window, the Velocity Lock status bit is cleared.	Velocity Control Units / Sec
472	Required – PV	Set		Velocity Standstill Window	REAL	This value establishes a window around zero speed. When the actual velocity is within this window the Velocity Standstill status bit is set. When actual speed falls outside this window, the Velocity Standstill status bit is cleared.	Velocity Control Units / Sec

**5-46.13.3 Acceleration Control Attributes**

The following attributes tables contain all accelerations related attributes associated with a Motion Axis Object instance.

**Set\*** - this attribute is generally updated via the cyclic command data set of the Controller-to-Device connection. When included as cyclic command data this attribute should not be updated via a Set service.

**Motion Axis Object, Class Code: 42<sub>hex</sub>****Table 5-46.32 Acceleration Control Signal Attributes**

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
480	Optional - C	Get		Acceleration Command	REAL	Command acceleration output from fine interpolator (if active) into acceleration loop when configured for acceleration control.	Accel Control Units / Sec <sup>2</sup>
481	Optional - C	Set*		Acceleration Trim	REAL	Additional acceleration command added to the acceleration loop summing junction.	Accel Control Units / Sec <sup>2</sup>
482	Optional - C	Get		Acceleration Reference	REAL	Command acceleration reference into acceleration loop summing junction.	Accel Control Units / Sec <sup>2</sup>
483	Required – NPVC Optional – T	Get		Acceleration Feedback	REAL	Actual acceleration of the axis based on the selected feedback device.	Accel Control Units / Sec <sup>2</sup>

**5-46.13.4 Torque/Force Reference Attributes**

The following attributes tables contain all torque/force related attributes associated with a Motion Axis Object instance.

Set\* - this attribute is generally updated via the cyclic command data set of the Controller-to-Device connection. When included as cyclic command data this attribute should not be updated via a Set service.

**Table 5-46.33 Torque/Force Reference Signal Attributes**

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
490	Required – C	Get		Torque Command	REAL	Command torque output from fine interpolator (if active) into torque input summing junction when configured for torque control.	% Motor Rated
491	Required – C	Set*		Torque Trim	REAL	Additional torque command added to the torque input summing junction.	% Motor Rated
492	Required – C	Get		Torque Reference	REAL	Commanded torque reference input signal before torque filter section representing the sum of the Torque Command and Torque Trim signal inputs.	% Motor Rated
493	Required – C	Get		Filtered Torque Reference	REAL	Commanded torque reference input signal after torque filter section.	% Motor Rated
494	Required – C	Get		Limited Torque Reference	REAL	Commanded torque reference input signal after torque limiter section.	% Motor Rated

Motion Axis Object, Class Code: 42<sub>hex</sub>

Table 5-46.34 Torque/Force Reference Configuration Attributes

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
496	Required – PVC	Set		Kj	REAL	Torque/Force Scaling Gain value that converts commanded acceleration into equivalent rated torque/force. Properly set, this value represents the total system inertia.	% Motor Rated / (Accel Units/Sec <sup>2</sup> )
498	Optional – C	Set		Friction Compensation	REAL	Value added to the current/torque command to offset the effects of coulomb friction.	% Motor Rated
502	Optional – C	Set		Torque LP Filter Bandwidth	REAL	Break frequency for the 2 <sup>nd</sup> order low pass filter applied to torque reference signal.	Radians/sec
503	Optional – C	Set		Torque Cmd Notch Freq.	REAL	Center frequency of the notch filter applied to the torque reference signal. A value of 0 for this attribute disabled this feature.	Radians/sec
504	Required – C	Set		Torque Limit - Positive	REAL	Determines the maximum positive torque that can be applied to the motor. If the drive attempts to exceed this value, the torque command is clamped to this value.	% Motor Rated
505	Required – C	Set		Torque Limit - Negative	REAL	Determines the maximum negative torque that can be applied to the motor. If the drive attempts to exceed this value, the torque command is clamped to this value.	% Motor Rated
506	Optional - C	Set		Torque Rate Limit	REAL	Limits the rate of change of the torque reference signal.	% Motor Rated / Sec
507	Optional – C	Set		Torque Threshold	REAL	Defines a maximum absolute torque level that, when exceeded, results in the Torque Threshold status bit being set.	% Motor Rated
508	Optional – C	Set		Overtorque Limit	REAL	Maximum limit for the Torque Reference signal that when exceeded for the duration specified by Overtorque Limit Time attribute, results in an Overtorque Limit exception. This feature allows the drive to generate an exception if there is a sudden increase in load torque during operation. This condition could occur if a bearing fails or there is some other mechanical failure.	% Motor Rated

Motion Axis Object, Class Code: 42<sub>hex</sub>

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
509	Optional – C	Set		Overtorque Limit Time	REAL	Specifies the amount of time allowed in an Overtorque Limit condition before generating an Overtorque Limit exception. A value of 0 for this attribute disabled the Undertorque feature.	Sec.
510	Optional – C	Set		Undertorque Limit	REAL	Minimum limit for the Torque Reference signal that when exceeded for the duration specified by Undertorque Limit Time attribute, results in an Undertorque Limit exception. This feature allows the drive to generate an exception if there is a sudden decrease in load torque during operation. This condition could occur if a load coupling breaks, tensioned web material breaks, etc.	% Motor Rated
511	Optional – C	Set		Undertorque Limit Time	REAL	Specifies the amount of time allowed in an Undertorque Limit condition before generating an Undertorque Limit exception. A value of 0 for this attribute disabled the Undertorque feature.	Sec.

**5-46.13.5 Current Loop Attributes**

The following attributes tables contain all current loop related attributes associated with a Motion Axis Object instance.

**Table 5-46.35 Current Loop Signal Attributes**

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
520	Required – C	Get		Iq Current Command	REAL	Represents the instantaneous value of the commanded torque producing current signal, Iq, prior to passing through the current limiter.	% Motor Rated
521	Optional – C	Get		Operative Current Limit	REAL	Represents the operative current limit based on multiple limit sources.	% Motor Rated

Motion Axis Object, Class Code: 42<sub>hex</sub>

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
522	Optional – C	Get		Current Limit Source	USINT	Represents the operative source of a torque limit should a current limit condition occur.	Enumeration: 0 = Not Limited 1 = Inverter Peak Current Limit 2 = Motor Peak Current Limit 3 = Inverter Thermal Current Limit 4 = Motor Thermal Current Limit 5 = Shunt Regulator Limit 6 – 127 = reserved 128 – 255 = vendor specific
523	Required – C	Get		Motor Electrical Angle	REAL	Calculated electrical angle of the motor based on motor pole count, commutation offset, and selected feedback device.	Degrees
524	Optional – C	Get		Iq Current Reference	REAL	Current reference signal, Iq, into the torque current loop summing junction.	% Motor Rated
525	Optional – C	Get		Id Current Reference	REAL	Command current reference, Id, into the flux current loop summing junction.	% Motor Rated
527	Optional – C	Get		Iq Current Error	REAL	Error between commanded and actual current that is the output of the torque producing, q-axis, current loop summing junction.	% Motor Rated
528	Optional – C	Get		Id Current Error	REAL	Error between commanded and actual current that is the output of the flux producing, d-axis, current loop summing junction.	% Motor Rated
529	Optional – C	Get		Iq Current Feedback	REAL	Actual torque current applied to the axis based on current sensor feedback.	% Motor Rated
530	Optional – C	Get		Id Current Feedback	REAL	Actual flux current applied to the axis based on current sensor feedback.	% Motor Rated
531	Optional – C	Get		Iq Decoupling	REAL	Signal added to the Iq control loop output to compensate for the effects of Id.	% Motor Rated
532	Optional – C	Get		Id Decoupling	REAL	Signal added to the Id control loop output to compensate for the effects of Iq.	% Motor Rated
533	Optional – C	Get		Vq Voltage Output	REAL	Instantaneous Torque/Force producing output voltage.	% Motor Rated

**Motion Axis Object, Class Code: 42<sub>hex</sub>**

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
534	Optional – C	Get		Vd Voltage Output	REAL	Instantaneous Flux producing output voltage.	% Motor Rated
535	Optional – C	Get		U Voltage Output	REAL	Instantaneous voltage applied to U phase	Volts
536	Optional – C	Get		V Voltage Output	REAL	Instantaneous voltage applied to V phase	Volts
537	Optional – C	Get		W Voltage Output	REAL	Instantaneous voltage applied to W phase	Volts
538	Optional – C	Get		U Current Feedback	REAL	Instantaneous current measured on U phase	Amps
539	Optional – C	Get		V Current Feedback	REAL	Instantaneous current measured on V phase	Amps
540	Optional – C	Get		W Current Feedback	REAL	Instantaneous current measured on W phase	Amps
541	Optional – C	Get		U Current Offset	REAL	Offset for U Phase current transducer.	Amps
542	Optional – C	Get		V Current Offset	REAL	Offset for V Phase current transducer.	Amps
543	Optional – C	Get		W Current Offset	REAL	Offset for W Phase current transducer.	Amps

**Table 5-46.36 Current Loop Configuration Attributes**

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
554	Optional – C	Get		Kqp	REAL	Iq Proportional Gain value that multiplies the Iq Current Error signal before applying it to the Iq decoupling summing junction.	Radians/Sec
555	Optional – C	Get		Kqi	REAL	Iq Integral Gain value that multiplies the Iq Current Error signal before applying it to the Iq Integrator Error accumulator.	Radians/Sec <sup>2</sup>
556	Optional – C	Get		Kdp	REAL	Id Proportional Gain value that multiplies the Id Current Error signal before applying it to the Iq decoupling summing junction.	Radians/Sec
557	Optional – C	Get		Kdi	REAL	Id Integral Gain value that multiplies the Id Current Error signal before applying it to the Id Integrator Error accumulator..	Radians/Sec <sup>2</sup>

Motion Axis Object, Class Code: 42<sub>hex</sub>

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
558	Optional – IM	Set		Flux Up Control	USINT	When the motion axis is enabled, DC current is applied to an induction motor to build stator flux before transitioning to the Running state. This attribute controls how an induction motor is to be fluxed in the Starting state prior to transitioning to the Running state. If No Delay is selected the axis transitions immediately to the Running state while the motor flux is building. With Manual Delay, the axis remains in the Starting state for the Flux Up Time to allow time for the motor to be fully fluxed. With Automatic Delay, the drive determines the amount of time to delay to fully flux the motor based on motor configuration attribute data or measurements.	Enumeration: 0 = No Delay 1 = Manual Delay 2 = Automatic Delay
559	Optional – IM	Set		Flux Up Time	REAL	Sets the amount of time the drive allows to build full motor flux before transitioning to the Running state.	Sec

**5-46.13.6 Frequency Control Attributes**

The following attributes tables contain all related attributes associated with the Frequency Control method of operation of a Motion Axis Object instance.

**Table 5-46.37 Frequency Control Signal Attributes**

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
570	Required - F	Get		Slip Compensation	REAL	Indicates the actual amount of slip compensation currently being applied.	RPM

**Table 5-46.38 Frequency Control Configuration Attributes**

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
575	Required - F	Set		Frequency Control Method	BYTE	The Basic Volts/Hertz control method applies voltage to the motor generally in direct proportion to the commanded frequency or speed.	Enumeration 0 = basic volts/hertz (R/F) 1-127 = reserved 128-255 = vendor specific

Motion Axis Object, Class Code: 42<sub>hex</sub>

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
571	Required - F	Set		Rated Slip Speed	REAL	Represents the amount of slip at motor rated current (full load) and motor rated frequency.	RPM
572	Required - F	Set		Maximum Voltage	REAL	Sets the highest phase-to-phase voltage the drive can output.	Volts (RMS)
573	Required - F	Set		Maximum Frequency	REAL	Sets the highest frequency the drive can output.	Hertz
574	Required - F	Set		Frequency Limit	REAL	Sets the frequency limit which is generally set above the Velocity Limit, translated to frequency, to allow for slip compensation.	Hertz
575	Required - F	Set		Break Voltage	REAL	Sets the phase-to-phase output voltage of the drive at the Break Frequency where boost ends.	Volts (RMS)
576	Required - F	Set		Break Frequency	REAL	Sets the output frequency of the drive at the Break Voltage where boost ends.	Hertz
577	Required - F	Set		Start Boost	REAL	Sets phase-to-phase voltage boost level for starting and accelerating.	Volts (RMS)
578	Required - F	Set		Run Boost	REAL	Sets the phase-to-phase voltage boost level for steady-state speed or deceleration.	Volts (RMS)

**5-46.13.7 Drive Output Attributes**

The following attributes tables contain all related attributes associated with the output of the drive inverter to the motor.

**Table 5-46.39 Drive Output Attributes**

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
600	Required - PVT	Get		Output Frequency	REAL	Time averaged output frequency applied to motor.	Hertz
601	Required - PVT	Get		Output Current	REAL	Total time averaged output current applied to motor.	Amps (RMS)
602	Required - PVT	Get		Output Voltage	REAL	Total time averaged phase-to-phase output voltage applied to motor.	Volts (RMS)
603	Required - PVT	Get		Output Power	REAL	Total time averaged output power of the motor. This value is based on the product of the Torque Reference signal and the Velocity Feedback.	Kilowatts

Motion Axis Object, Class Code: 42<sub>hex</sub>

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
604	Optional - PVT	Set		PWM Frequency	UINT	Sets the carrier frequency for the Pulse Width Modulation output to the motor. Drive derating is required at higher PWM frequencies due to switching losses. Current loop update time is tied directly to the PWM frequency so loop performance generally increases with increasing PWM rate.	Hertz

**5-46.14 Stopping & Braking Attributes**

The following attributes tables contain all active stopping and braking related attributes associated with a Motion Axis Object instance.

**Table 5-46.40 Stopping/Braking Attributes**

Attr ID	Need in Implem	Access Rule	NV	Attribute Name	Data Type	Description of Attribute	Semantics of Values
610	Required -PVT	Set		Stopping Mode	USINT	When initiating a drive disable or a fault, this value determines the stopping method to apply to the motor. The final state after the stopping method is applied is the Stopped state. In the Stopped state the drive's inverter power structure should either be Disabled (Disable selection) and free of torque or actively held (Hold selection) in a static condition via servo action.	See Semantics
611	Required - PVT	Set		Stopping Current	REAL	When initiating a drive disable, this value determines the maximum amount of torque producing current available to stop the motor when the Stopping Mode is set to Current Decel. If this attribute is not supported, the drive device shall use the configured Positive and Negative Peak Current Limits.	% Motor Rated
612	Required - PVT	Set		Stopping Time Limit	REAL	When disabling or aborting an axis, this parameter determines the maximum amount of time Stopping Current can be applied to bring the motor to a successful stop before the axis is disabled and all current is removed from the motor.	Sec

Motion Axis Object, Class Code: 42<sub>hex</sub>

Attr ID	Need in Implem	Access Rule	NV	Attribute Name	Data Type	Description of Attribute	Semantics of Values
613	Optional – PVT (PM Motors Only)	Set		Resistive Brake Contact Delay	REAL	When an external resistive brake is used, the Resistive Brake Contact Delay can be set to delay the enabling of the drive power structure until after the resistive brake has had time to connect the motor to the drive. See Semantics for details.	Sec
614	Optional - PVT	Set		Mechanical Brake Control	USINT	This attribute allows direct control of the drive output connected to the mechanical brake mechanism. When set to release or engage the brake, the brake is released or engaged without regard to the current state of the axis. An axis state transition however can override the Mechanical Brake Control setting. For example, if this attribute is a 0 and the brake is released and then the axis transitions the Stopped state to the Running state and back to the Stopped state, the brake automatically engages as part of the transition to the stopped state. The device indicates this condition by changing the Mechanical Brake Control setting to 1, indicating that the brake is engaged.	Enumeration: 0 = Brake Release 1 = Brake Engage
615	Required - PVT	Set		Mechanical Brake Release Delay	REAL	When enabling the axis, the Mechanical Brake Release Delay value determines the amount of time the drive shall delay transition from the Starting state to the Running or Testing states. This delay prevents any commanded motion of the motion axis until the external mechanical brake has had enough time to disengage. See Semantics for details.	Sec

Motion Axis Object, Class Code: 42<sub>hex</sub>

Attr ID	Need in Implem	Access Rule	NV	Attribute Name	Data Type	Description of Attribute	Semantics of Values
616	Required - PVT	Set		Mechanical Brake Engage Delay	REAL	When disabling the axis, the Mechanical Brake Engage Delay value determines the amount of time the inverter power structure shall remain enabled after the axis has decelerated to standstill. This allows time for an external mechanical brake to engage. The configured Stopping Mode determines the type of stopping sequence applied .See Semantics for details.	Sec

**5-46.14.1 Semantics****5-46.14.2 Attribute #610 – Stopping Mode**

When disabling or aborting an axis, this attribute determines the stopping method to apply to the motor. The final state after the stopping method is applied is the Stopped state. In the Stopped state the axis' inverter power structure should either be Disabled (Disable selection) and free of torque or actively held (Hold selection) in a static condition via servo action. This attribute has no impact or relationship to the planner generated acceleration and deceleration profiles.

**Table 5-46.41 Stopping Mode Enumeration Definitions**

Enum.	Req. Opt.	Name	Description
0	R/CF	Disabled & Coast	Disable & Coast immediately disables the inverter power structure which causes the motor to coast unless some form of external braking is applied. This is equivalent to an IEC-60204-1Category 0 Stop.
1	R/C	Current Decel & Disable	Current Decel & Disable forces the drive to stop under control of the configured Stopping Current. This is accomplished by forcing the relevant upstream control loops to saturate by setting the appropriate command references to fixed values. In the position mode, the position command is reset to actual position when the axis reaches zero velocity. Once stopped, or the configured Stopping Time limit expires, the device disables the power structure and control loops. This stop mode complies with the IEC-60204-1 Category 1 Stop.
2	R/V	Ramped Decel & Disable	Ramped Decel & Disable applies only to the velocity control mode. This stop mode immediately zeroes the output of the Velocity Command Selector causing the Accel Limiter to ramp the motor speed down to zero. Once stopped, or the configured Stopping Time limit expires, the device disables the power structure and control loops. This stop mode also complies with the IEC-60204-1 Category 1 Stop.
3	O/C	Current Decel & Hold	Current Decel & Hold behaves like Current Decel & Disable, but leaves the power structure and control loops active under local command reference control with holding torque. In this way there is active holding torque to maintain the stopped condition. This stop mode complies with the IEC-60204-1 Category 2 Stop.

Motion Axis Object, Class Code: 42<sub>hex</sub>

Enum.	Req. Opt.	Name	Description
4	O/F	Ramped Decel & Hold	Ramped Decel & Hold behaves like Ramped Decel & Disable, but leaves the power structure and control loops active under local command reference control. In this way there is active holding torque to maintain the stopped condition. This stop mode also complies with the IEC-60204-1 Category 2 Stop.
5-127		Reserved	
128-255		Vendor Specific	

**5-46.14.3 Attribute #613 – Resistive Brake Contact Delay**

When an external resistive brake is used, an external contactor switches the UVW motor leads from the inverter power structure to an energy dissipating resistor to stop the motor. This switching does not occur instantaneously and so enabling the power structure too early can cause electrical arcing across the contactor. To prevent this condition, the Resistive Brake Contact Delay can be set to the maximum time that it takes to fully close the contactor across the UVW motor lines so when the axis is enabled, the inverter power structure is not enabled until after the Resistive Brake Contact Delay Time has expired. Resistive Brake operation is only applicable to PM Motor types. The following sequence illustrates how the Resistive Brake Contact Delay factors into the overall Enable Sequence that may also include the operation of a Mechanical Brake.

**Drive Enable Sequence:**

1. Switch to Starting state.
2. Activate Resistive Brake contactor to connect motor to drive.
3. Wait for “Resistive Brake Contact Delay” while Resistive Brake contacts close.
4. Enable drive power structure.
5. Activate Mechanical Brake output to release brake.
6. Wait for “Mechanical Brake Release Delay” while brake releases.
7. Transition to Running state.

**5-46.14.4 Attribute #615 – Mechanical Brake Release Delay**

When enabling the axis, the Mechanical Brake Release Delay value determines the amount of time the axis shall delay transition from the Starting state to the Running or Testing states. This delay prevents any commanded motion of the axis until the external mechanical brake has had enough time to disengage.

**Drive Enable Sequence:**

1. Switch to Starting state.
2. Activate Resistive Brake contactor to connect motor to drive.
3. Wait for “Resistive Brake Contact Delay” while Resistive Brake contacts close.
4. Enable drive power structure.
5. Activate Mechanical Brake output to release brake.
6. Wait for “Mechanical Brake Release Delay” while brake releases.
7. Transition to Running (or Testing) state.

#### 5-46.14.5 Attribute #616 – Mechanical Brake Engage Delay

When disabling the axis, the Mechanical Brake Engage Delay value determines the amount of time the inverter power structure shall remain enabled after the axis has decelerated to standstill. This allows time for an external mechanical brake to engage. The configured Stopping Mode determines the type of stopping sequence applied.

##### Drive Disable Sequence (Category 0 Stop):

1. Disable drive power structure.
2. Transition to Stopped state.
3. Deactivate Mechanical Brake output to engage brake.
4. Deactivate Resistive Brake contactor to disconnect motor from inverter power structure.

##### Drive Disable Sequence (Category 1 Stop):

1. Switch to Stopping state.
2. Apply “Current Decel” or “Ramp Decel” method to stop motor.
3. Wait for zero speed or “Stopping Time Limit”, whichever occurs first.
4. Deactivate Mechanical Brake output to engage brake.
5. Wait for “Mechanical Brake Engage Delay” while brake engages.
6. Disable inverter power structure.
7. Transition to Stopped state.
8. Deactivate Resistive Brake contactor to disconnect motor from inverter power structure.

##### Drive Disable Sequence (Category 2 Stop): The mechanical brake not used.

#### 5-46.15 DC Bus Control Attributes

The following table contains Motion Axis Object instance attributes associated with DC Bus control including functionality to address both under-voltage and over-voltage conditions.

**Table 5-46.42 DC Bus Control Attributes**

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
620	Required - PVT	Get		DC Bus Voltage	REAL	Measured DC Bus Voltage.	Volts
621	Required - PVT	Get		DC Bus Voltage - Nominal	REAL	Normal DC Bus Voltage during operation as determined by averaging the DC Bus Voltage over a drive specific time interval. This value is used as the basis for Bus Overvoltage and Undervoltage limits. NOTE: If the drive does not support this bus voltage averaging concept, the alternative is to hard code this value.	Volts

Motion Axis Object, Class Code: 42<sub>hex</sub>

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
624	Required - PVT	Set		Bus Regulator Action	USINT	Controls the method of operation of the DC Bus Regulator that addresses the regenerative over-voltage conditions that can occur when decelerating a motor.  If Disabled, no regulation is applied to the DC Bus level to control regenerative energy sourced by the motor.  When Shunt Regulator is selected the associated shunt regulation hardware is applied to the DC Bus to dissipate regenerative energy via a resistor.	Enumeration: 0 = Disabled (R/PVT) 1 = Shunt Regulator (R/PVT) 2-127 = reserved 128-255 = vendor specific
625	Optional - PVT	Set		Bus Regulator Power Limit	REAL	Limits the amount of power allowed to transfer between the motor and the DC Bus during regenerative braking of the motor load. When using an external shunt resistor, set this value to its maximum value.	% Regulator Rated
627	Optional - PVT	Set		Power Loss Action	USINT	Set the reaction to a DC Bus under-voltage condition when the DC Bus drops below a hard-coded threshold in the drive or the configured Power Loss Threshold	Enumeration: 0 = Disabled & Coast 1 = Decel 2 = Continue 3-127 = reserved 128-255 = vendor specific
628	Optional – PVT	Set		Power Loss Threshold	REAL	Sets the Level for Power Loss as percent of nominal DC Bus Voltage. If this value is 0, the hard-coded threshold is used.	%
629	Optional - PVT	Set		Drive Shutdown Action	UINT	Selects the drive's action when transitioning to the Shutdown state. Standard action is to immediately disable the drive's power structure. Alternatively, action can be taken to drop the DC Bus voltage as well.	Enumeration: 0 = Disable (R) 1 = Drop DC Bus (O) 2-127 = reserved 128-255 = vendor specific

### 5-46.16 Power and Thermal Management Attributes

The following attribute tables contain all power and thermal related attributes associated with a Motion Axis Object instance.

**Table 5-46.43 Power and Thermal Management Status Attributes**

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
635	Required - PVT	Get		Motor Capacity	REAL	Real-time estimate of the continuous rated motor thermal capacity utilized during operation based on the motor thermal model. A value of 100% would indicate that the motor is being used at 100% of rated capacity as determined by the continuous current rating of the motor.	% Motor Rated
636	Required - PVT	Get		Inverter Capacity	REAL	Real-time estimate of the continuous rated inverter thermal capacity utilized during operation based on the inverter thermal model. A value of 100% would indicate that the inverter is being used at 100% of rated capacity as determined by the continuous current rating of the inverter.	% Inverter Rated
637	Optional - PVT	Get		Converter Capacity	REAL	Real-time estimate of the continuous rated converter thermal capacity utilized during operation based on the converter thermal model. A value of 100% would indicate that the converter is being used at 100% of rated capacity as determined by the continuous current rating of the converter.	% Converter Rated
638	Optional - PVT	Get		Bus Regulator Capacity	REAL	Real-time estimate of the continuous rated bus regulator thermal capacity utilized during operation based on the bus regulator thermal model. A value of 100% would indicate that the bus regulator is being used at 100% of rated capacity as determined by the continuous current rating of the bus regulator.	% Regulator Rated
639	Optional - PVT	Get		Drive Ambient Temperature	REAL	Current internal ambient temperature of the device enclosure.	Degrees C
640	Optional - PVT	Get		Inverter Heatsink Temperature	REAL	Current temperature of the drive inverter heatsink, typically based on an embedded temp sensor.	Degrees C

Motion Axis Object, Class Code: 42<sub>hex</sub>

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
641	Optional - PVT	Get		Inverter Temperature	REAL	Current temperature of the power block used in the inverter's power structure, sometime referred to as the semiconductor junction temperature.	Degrees C
642	Optional - PVT	Get		Motor Temperature	REAL	Current temperature of the motor stator, typically based on an embedded temp sensor.	Degrees C
643	Optional - PVT	Get		Feedback 1 Temperature	REAL	Current temperature of the Feedback 1 device.	Degrees C
644	Optional - PVT	Get		Feedback 2 Temperature	REAL	Current temperature of the Feedback 2 device.	Degrees C
645	Optional – PVT	Get		Inverter Overload Limit	REAL	Factory set maximum limit for Inverter Capacity that when exceeded triggers the selected Inverter Overload action.	% Inverter Rated

Table 5-46.44 Power and Thermal Management Configuration Attributes

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
646	Optional - PVT	Set		Motor Overload Action	USINT	Selects the device's response to a motor overload condition based on an $I^2t$ based motor thermal model. The motor overload condition occurs when the motor thermal model indicates that the Motor Capacity has exceeded the Motor Overload Limit. This motor overload action functionality is independent of the motor overload exception action functionality.  No explicit action is taken by the device in the overload condition if None is the selected overload action. Selecting the Current Foldback action, however, results in a reduction of the current command via the Motor Thermal Current Limit attribute value that is reduced in proportion to the percentage difference between Motor Capacity and the Motor Overload Limit.	Enumeration: 0 = None (R) 1 = Current Foldback (O) 2-127 = reserved 128-255 = vendor specific

Motion Axis Object, Class Code: 42<sub>hex</sub>

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
647	Optional - PVT	Set		Inverter Overload Action	USINT	Selects the device's response to an inverter overload alarm condition based on an I <sup>2</sup> t based inverter thermal model. The inverter overload alarm condition occurs when the inverter thermal model indicates that the Inverter Capacity has exceeded the Inverter Overload Limit. The Inverter Overload Action provides opportunities to mitigate the overload condition without stopping operation. This inverter overload action functionality is independent of the motor overload exception action functionality.  No explicit action is taken by the device in the overload condition if None is the selected overload action. Selecting the Current Foldback action, however, results in a reduction of the current command via the Inverter Thermal Current Limit attribute value that is reduced in proportion to the percentage difference between Inverter Capacity and the Inverter Overload Limit	Enumeration: 0 = None (R) 1 = Current Foldback (O) 2-127 = reserved 128-255 = vendor specific

**5-46.17 Axis Status Attributes**

The following attributes tables contain all status attributes associated with a Motion Axis Object instance.

**Table 5-46.45 Axis Status Attributes**

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
650	Required - All	Get		Axis State	USINT	Enumerated value indicating the state of the axis.	Enumeration: 0 = Initializing 1 = Pre-Charge 2 = Stopped 3 = Starting 4 = Running 5 = Testing 6 = Stopping 7 = Aborting 8 = Major Faulted 9 = Start Inhibited 10 = Shutdown

Motion Axis Object, Class Code: 42<sub>hex</sub>

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
							11-255: Reserved
651	Required - All	Get		Axis Status	DWORD	Collection of bits indicating the internal status of the motion axis.	See Semantics
652	Required - All	Get		Axis Status - Mfg	DWORD	Collection of bits indicating the status of the axis.	Bitmap: 0-31: Vendor Specific (Published in Product Manual)
653	Required - All	Get		Axis I/O Status	DWORD	The Axis I/O Status attribute is a 32-bit collection of bits indicating the state of standard digital inputs and outputs associated with the operation of this axis. A value of zero for a given input bit indicates a logical 0 value, while a value of 1 indicates a logical 1 value	See Semantics
654	Required - All	Get		Axis I/O Status - Mfg	DWORD	Collection of bits indicating the state of vendor specific digital inputs associated with the operation of this axis. A value of zero for a given input bit indicates a logical 0 value, while a value of 1 indicates a logical 1 value.	Bitmap: 0-31: Vendor Specific (Published in Product Manual)
94	Required – All	Set*		Status Data Set	BYTE	Bit mapped field that determines what status data values are to be transmitted to the controller via the Device-to-Controller Connection.	See Semantics

**5-46.17.1 Semantics****5-46.17.2 Attribute #651 – Axis Status**

The Axis Status attribute is a 32-bit collection of bits indicating various internal status conditions of the motion axis.

**Table 5-46.46 Axis Status Bit Definitions**

Bit	Rec. Opt.	Name	Description
0	R	Local Control	This bit is set if axis is taking command reference and services from local interface instead of via the remote interface (e.g.: the CIP Motion controller).
1	R	Alarm	This bit is set if the axis has detected one or more exception conditions configured to generate an alarm.
2	R	DC Bus Up	This bit is set if DC Bus has charged up to an operational voltage level.
3	R	Power Structure Enabled	This bit is set if the axis amplifier is energized and capable of generating motor flux and torque.
4	IM	Motor Flux Up	This bit is set if motor flux for an induction motor has reached an operational level.
5	R	Tracking Command	This bit is set if the axis control structure is now actively tracking the

**Motion Axis Object, Class Code: 42<sub>hex</sub>**

<b>Bit</b>	<b>Rec. Opt.</b>	<b>Name</b>	<b>Description</b>
			command reference from motion planner
6	R/P	Position Lock	This bit is set if the actual position is within Position Lock Tolerance of command position.
7	R/V	Velocity Lock	This bit is set if the actual velocity is within Velocity Lock Tolerance of command velocity.
8	R	Velocity Standstill	This bit is set if the actual velocity is within Velocity Standstill Window of 0.
9	R	Velocity Threshold	This bit is set if the absolute actual velocity is below Velocity Threshold.
10	R/V	Velocity Limit	This bit is set if the command velocity in velocity control mode is currently being limited by the Velocity Limiter.
11	R/V	Acceleration Limit	This bit is set if the rate of change of the command velocity in velocity control mode is currently being limited by the Acceleration Limiter.
12	R/V	Deceleration Limit	This bit is set if the rate of change of command acceleration in velocity control mode is currently being limited by the Deceleration Limiter.
13	R/C	Torque Threshold	This bit is set if the absolute filtered torque reference is below the Torque Threshold.
14	R/C	Current Limit	This bit is set if the command current, Iq, is currently being limited by the Current Vector Limiter.
15	R/C	Thermal Limit	This bit is set if Current Limit condition of the axis is being limited by any of the axis's Thermal Models.
16-31		Reserved	

**5-46.17.3 Attribute #653 – Axis I/O Status**

The Axis I/O Status attribute is a 32-bit collection of bits indicating the state of standard digital inputs and outputs associated with the operation of this axis. A value of zero for a given input bit indicates a logical 0 value, while a value of 1 indicates a logical 1 value.

**Table 5-46.47 Axis I/O Status Bit Definitions**

<b>Bit</b>	<b>Rec. Opt.</b>	<b>Name</b>	<b>Description</b>
0	R	Enable Input	
1	R/P	Home Input	
2	R/P	Registration 1 Input	
3	O/P	Registration 2 Input	
4	R/P	Positive Overtravel Input	
5	R/P	Negative Overtravel Input	
6	O	Feedback 1 Thermostat	
7	O	Feedback 1 Hall S1	
8	O	Feedback 1 Hall S2	
9	O	Feedback 1 Hall S3	
10	O	Feedback 1 Ch A	
11	O	Feedback 1 Ch B	
12	O	Feedback 2 Ch A	
13	O	Feedback 2 Ch B	
14	O	Feedback 3 Ch A	

**Motion Axis Object, Class Code: 42<sub>hex</sub>**

Bit	Rec. Opt.	Name	Description
15	O	Feedback 3 Ch B	
16	O	Feedback 4 Ch A	
17	O	Feedback 4 Ch B	
18	O	Resistive Brake Output	
19	O	Mechanical Brake Output	
20-31		Reserved.	

**5-46.17.4 Attribute #94 – Status Data Set**

The Status Data Set attribute is an 8-bit collection of bits indicating what Status attributes are to be transmitted to the controller over the Device-to-Controller Connection. Status data appears in connection data structure in the same order as the bit numbers defined in the table below. For example, Axis Fault Status would appear before, Axis Alarm Status data in the Status Data Set structure. Multiple attributes comprising a selected Status Data Element are transmitted in the order listed from top to bottom, so the Axis Fault Code is transmitted before Mfg Axis Fault Code.

**Table 5-46.48 Status Data Set Bit Definitions**

Bit	Status Data Element Produced	Data Type
0	Initialization Fault Code Initialization Fault Code - Mfg Axis Fault Code Axis Fault Code - Mfg Axis Alarm Code Axis Alarm Code - Mfg Start Inhibit Code Start Inhibit Code - Mfg Axis Fault Time Stamp	USINT USINT USINT USINT USINT USINT USINT USINT LINT
1	Axis Fault Status Axis Fault Status - Mfg	LWORD LWORD
2	Axis Alarm Status Axis Alarm Status - Mfg	LWORD LWORD
3	Start Inhibit Status Start Inhibit Status - Mfg	WORD WORD
4	Axis Status Axis Status - Mfg Axis I/O Status Axis I/O Status - Mfg	DWORD DWORD DWORD DWORD
5	Reserved	Reserved
6	Reserved	Reserved
7	Vendor Specific	Vendor Specific

**5-46.18 Exception, Fault, and Alarm Status Attributes**

The following attribute tables contain all exception, fault, and alarm related attributes associated with a Motion Axis Object instance. Exceptions are conditions that can occur during motion axis operation that have the potential of generating faults or alarms.

Motion Axis Object, Class Code: 42<sub>hex</sub>

Table 5-46.49 Exception, Fault and Alarm Status Attributes

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
655	Required - All	Get		Axis Exception Status	LWORD	A bit map that represents the current state of all standard exception conditions. See the Std. Exception Table bit map definition later in this section for detail on the bit locations. Each exception has a corresponding Exception Action. Exceptions that are configured to be Ignored are only visible in this attribute.	See Semantics.
656	Required - All	Get		Axis Exception Status - Mfg	LWORD	A bit map that represents the current state of all manufacturer specific exception conditions. See the Mfg. Exception Table published in drive product manual. Each exception has a corresponding Exception Action. Exceptions that are configured to be Ignored are only visible in this attribute.	See Mfg. Exception Table (Published in Product Manual)
657	Required - All	Get		Axis Fault Status	LWORD	A bit map that represents the state of all standard runtime faults. The bit map is identical to that of the Axis Exception Status attribute. Fault status bits when set are latched until a fault reset occurs. A fault reset clears the runtime fault bits, but the bits set again immediately if the underlying exception condition is still present. Any exceptions whose Exception Action is configured to ignore or report as alarms do not appear in this attribute.	See Semantics
658	Required - All	Get		Axis Fault Status - Mfg	LWORD	A bit map that represents the state of all manufacturer specific runtime faults. The bit map is identical to that of the Mfg. Axis Exception Status attribute. Fault status bits when set are latched until a fault reset occurs. A fault reset clears the runtime fault bits, but the bits set again immediately if the underlying exception condition is still present. Any exceptions whose Exception Action is configured to ignore or report as alarms do not appear in this attribute.	See Mfg. Exception Table (Published in Product Manual)

Motion Axis Object, Class Code: 42<sub>hex</sub>

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
659	Required - All	Get		Axis Alarm Status	LWORD	A bit map that represents the current state of all standard alarm conditions. The bit map is identical to that of the Std. Axis Exception Status attribute. Only exception conditions whose Exception Action is configured to report as an alarm appear in this attribute, and will not be reported in the Axis Fault Status attribute. Alarm status bits when set are not latched and will clear as soon as the underlying exception condition is corrected.	See Semantics
660	Required - All	Get		Axis Alarm Status - Mfg	LWORD	A bit map that represents the current state of all manufacturer specific alarm conditions. The bit map is identical to that of the Mfg. Axis Exception Status attribute. Only exception conditions whose Exception Action is configured to report as an alarm appear in this attribute, and will not be reported in the Axis Fault Status attribute. Alarm status bits when set are not latched and will clear as soon as the underlying exception condition is corrected.	See Mfg. Exception Table. (Published in Product Manual)
661	Required - All	Get		Axis Fault Code	USINT	An 8-bit enumeration that specifies the standard exception that led to the current faulted state. The value corresponds to the bit position of the exception condition, and a value of 0 indicates that no standard runtime faults currently exist. This attribute only reports the first runtime fault that transitioned the axis from an unfaulted state to the faulted state, and then only if that was not a Mfg. Axis Fault. Subsequent runtime faults that occur when the Axis Fault Code attribute or the Mfg. Axis Fault Code attribute is already nonzero are not reported in this attribute. Note that no runtime faults exist if both the Axis Fault Code and the Mfg. Axis Fault Code are zero. The Axis Fault Code and the Mfg. Axis Fault Code can never both be non-zero at the same time.	See Semantics

Motion Axis Object, Class Code: 42<sub>hex</sub>

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
662	Required - All	Get		Axis Fault Code - Mfg	USINT	An 8-bit enumeration that specifies the manufacturer specific exception that led to the current faulted state. The value corresponds to the bit position of the exception condition, and a value of 0 indicates that no manufacturer runtime faults currently exist. This attribute only reports the first runtime fault that transitioned the axis from an unfaulted state to the faulted state, and then only if that was a Mfg. Axis Fault. Subsequent runtime faults that occur when either the Axis Fault Code attribute or the Mfg. Axis Fault Code attribute is already nonzero are not reported in this attribute. Note that no runtime faults exist if both the Axis Fault Code and the Mfg. Axis Fault Code are zero. The Axis Fault Code and the Mfg. Axis Fault Code can never both be non-zero at the same time.	See Mfg. Exception Table (Published in Product Manual)
663	Required - All	Get		Axis Fault Time Stamp	LINT	A 64-bit Time Stamps of the exception that led to the current non-zero Axis Fault Code or Mfg Axis Fault Code. The time stamp format is absolute follows the CIP Sync standard with 0 corresponding to January 1, 1970.	Nanoseconds
664	Required - All	Get		Axis Alarm Code	USINT	An 8-bit enumeration that specifies the exception that led to the current standard alarm condition. The value corresponds to the bit position of the exception condition that led to the alarm, and a value of 0 indicates that no standard alarms currently are active. If the standard alarm condition is removed, this attribute will automatically be cleared or else be set to another standard alarm condition if one is present. If multiple standard alarm conditions are present, the Axis Alarm Code is set to the lowest active alarm bit number. The Axis Alarm Code and the Mfg. Axis Alarm Code can both be non-zero at the same time.	See Semantics

Motion Axis Object, Class Code: 42<sub>hex</sub>

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
665	Required - All	Get		Axis Alarm Code - Mfg	USINT	An 8-bit enumeration that specifies the manufacturer specific exception that led to the current alarm condition. The value corresponds to the bit position of the manufacturer exception condition that led to the alarm, and a value of 0 indicates that no manufacturer alarms currently exist. If the manufacturer alarm condition is removed, this attribute will automatically be cleared or else set to another manufacturer alarm condition if one is present. If multiple manufacturer alarm conditions are present, the Mfg. Axis Alarm Code is set to the lowest active alarm bit number. The Axis Alarm Code and the Mfg. Axis Alarm Code can both be non-zero at the same time.	See Mfg. Exception Table (Published in Product Manual)
666	Optional - All	Get		Fault Sub Code	USINT	A 16-bit enumeration that specifies the fault sub code for the fault condition currently displayed on the front panel of the device (and reported in the Axis Fault Code). This attribute allows additional detail to be provided in the case of some faults, and has a different interpretation for each fault. Non-zero values indicate that additional detail is present.	Enumeration specific to fault condition.

**5-46.18.1 Semantics****5-46.18.2 Standard Exception Table**

The following table defines a list of standard exceptions associated with the Axis Exception Status attributes but also applicable to Axis Fault Status, Axis Fault Code, Axis Alarm Status, and Axis Alarm Code attributes.

**Table 5-46.50 Standard Exception Table**

Bit	Exception	Description
0	Reserved	This bit cannot be used since the Alarm Codes and Fault Code are defined by the associated exception bit number and an Alarm Code or Fault Code of 0 means no alarm or fault condition is present.
1	Motor Stall	Induction motor stall condition detected.
2	Motor Commutation	Permanent magnet motor commutation problem detected. Example would be an illegal state “111” or “000” for Hall Sensor.
4	Motor Overspeed FL	Motor speed has exceeded its maximum limit given by either Rotary Motor Max Speed or Linear Motor Max Speed attributes depending on the type of motor.

Motion Axis Object, Class Code: 42<sub>hex</sub>

Bit	Exception	Description
5	Motor Overspeed UL	Motor speed has exceeded the user defined Motor Overspeed User Limit margin above the Positive Velocity Limit or below the Negative Velocity Limit.
6	Motor Overtemperature FL	Motor temperature has exceeded its factory set temperature limit given by Motor Overtemperature Factory Limit, or the integral motor thermal switch has tripped.
7	Motor Overtemperature UL	Motor temperature has exceeded the user defined temperature limit given by Motor Overtemperature User Limit.
8	Motor Thermal Overload FL	Motor thermal model has exceeded its factory set thermal capacity limit given by Motor Thermal Overload Factory Limit.
9	Motor Thermal Overload UL	Motor thermal model has exceeded its user defined thermal capacity given by Motor Thermal Overload User Limit.
10	Reserved	
11	Reserved	
12	Inverter Overcurrent	Inverter current has exceeded the factory set peak or instantaneous current limit.
13	Inverter Overtemperature FL	Inverter temperature has exceeded its factory set temperature limit given by Inverter Overtemperature Factory Limit.
14	Inverter Overtemperature UL	Inverter temperature has exceeded the user defined temperature limit given by Inverter Overtemperature User Limit.
15	Inverter Thermal Overload FL	Inverter thermal model has exceeded its factory set thermal capacity limit given by Inverter Thermal Overload Factory Limit.
16	Inverter Thermal Overload UL	Inverter thermal model has exceeded its user defined thermal capacity given by Inverter Thermal Overload User Limit.
17	Reserved	
18	Reserved	
19	Converter Overtemperature FL	Converter temperature has exceeded its factory set temperature limit given by Converter Overtemperature Factory Limit.
20	Converter Overtemperature UL	Converter temperature has exceeded the user defined temperature limit given by Converter Overtemperature User Limit.
21	Converter Thermal Overload FL	Converter thermal model has exceeded its factory set thermal capacity limit given by Converter Thermal Overload Factory Limit.
22	Converter Thermal Overload UL	Converter thermal model has exceeded its user defined thermal capacity given by Converter Thermal Overload User Limit.
23	Converter AC Phase Loss	One or more phases have been lost on the AC line to the converter.
24	Converter AC Ground Current	Excessive Ground Current was detected by the converter.
25	Reserved	
26	Reserved	
27	Bus Regulator Overtemperature FL	Bus Regulator temperature has exceeded its factory set temperature limit given by Bus Regulator Overtemperature Factory Limit.
28	Bus Regulator Overtemperature UL	Bus Regulator temperature has exceeded the user defined temperature limit given by Bus Regulator Overtemperature User Limit.
29	Bus Regulator Thermal Overload FL	Bus Regulator thermal model has exceeded its factory set thermal capacity limit given by Bus Regulator Thermal Overload Factory Limit.
30	Bus Regulator Thermal Overload UL	Bus Regulator thermal model has exceeded its user defined thermal capacity given by Bus Regulator Thermal Overload User Limit.
31	Bus Undervoltage FL	DC Bus voltage level is below the factory set limit given by Bus Undervoltage Factory Limit.
32	Bus Undervoltage UL	DC Bus voltage level is below user defined limit given by Bus Undervoltage User Limit.

Motion Axis Object, Class Code: 42<sub>hex</sub>

Bit	Exception	Description
33	Bus Overvoltage FL	DC Bus voltage level is above the factory set limit given by Bus Overvoltage Factory Limit.
34	Bus Overvoltage UL	DC Bus voltage level is above user defined limit given by Bus Overvoltage User Limit.
35	Reserved	
36	Reserved	
37	Feedback 1 Signal Noise	Noise induced A/B channel state changes were detected on the Feedback 1 signal connection. Specifically, the rate at which these noise events have occurred has exceeded a factory set rate limit.
38	Feedback 1 Signal Loss	One or more A/B channel Feedback 1 device signal connections are open, shorted, missing, or severely attenuated. Specifically, the detected voltage levels of the signals are below a factory set limit.
39	Feedback 1 Serial Communication	The number of consecutive missed or corrupted serial data packets over the serial data channel associated with Feedback 1 has exceeded a factory set limit.
40	Feedback 1 Power	A problem has been found related to the power supply, or battery, associated with the Feedback 1 device.
41	Reserved	
43	Reserved	
44	Feedback 2 Signal Noise	Noise induced A/B channel state changes were detected on the Feedback 2 signal connection. Specifically, the rate at which these noise events have occurred has exceeded a factory set rate limit.
45	Feedback 2 Signal Loss	One or more A/B channel Feedback 2 device signal connections are open, shorted, missing, or severely attenuated. Specifically, the detected voltage levels of the signals are below a factory set limit.
46	Feedback 2 Serial Communication	The number of consecutive missed or corrupted serial data packets over the serial data channel associated with Feedback 2 has exceeded a factory set limit.
47	Feedback 2 Power	A problem has been found related to the power supply, or battery, associated with the Feedback 2 device.
48	Reserved	
49	Reserved	
50	Hardware Overtravel Positive	Axis moved beyond the physical travel limits in the positive direction and activated the Positive Overtravel limit switch.
51	Hardware Overtravel Negative	Axis moved beyond the physical travel limits in the negative direction and activated the Negative Overtravel limit switch.
52	Reserved	
53	Reserved	
54	Excessive Position Error	The Position Error value of the position control loop has exceeded the configured value for Position Error Tolerance.
55	Excessive Velocity Error	The Velocity Error value of the velocity control loop has exceeded the configured value for Velocity Error Tolerance.
56	Overtorque Limit	Motor torque has risen above user defined maximum torque level given by Overtorque Limit.
57	Undertorque Limit	Motor torque has dropped below user defined minimum torque level given by Undertorque Limit.
58	Reserved	
59	Reserved	
60	Illegal Mode Change	Controller has specified an unsupported Control Mode or Feedback Configuration

**Motion Axis Object, Class Code: 42<sub>hex</sub>**

<b>Bit</b>	<b>Exception</b>	<b>Description</b>
61	Enable Input Deactivated	Enable has been deactivated while the axis is in Running state.
62	Controller Initiated Exception	Exception generated specifically by controller.
63	External Exception Input	Exception generated by external input to device.

**5-46.19    Exception Limit Attributes**

The following attribute tables contain all exception limit related attributes associated with a Motion Axis Object instance. Exception Limit attributes define the conditions under which a corresponding exception is generated during motion axis operation that has the potential of generating either a fault or alarm. They are typically associated with temperature, current, and voltage conditions of the device that are continuous in nature. Factory Limits (FL) for exceptions are usually hard coded in the device and typically result in a major fault condition. User Limits (UL) for exceptions are configurable and typically used to generate a minor fault, or alarm condition. For this reason, the User Limits are generally set inside the corresponding Factory Limits. Note that the triggering of a User Limit exception does not preclude triggering of the corresponding Factory Limit exception; the two exception trigger conditions are totally independent of one another.

**Table 5-46.51 Exception Factory Limit Info Attributes**

<b>Attr ID</b>	<b>Need in Implem</b>	<b>Access Rule</b>	<b>N V</b>	<b>Attribute Name</b>	<b>Data Type</b>	<b>Description of Attribute</b>	<b>Semantics of Values</b>
680	Optional - PVT	Get		Motor Overtemperature Factory Limit	REAL	Returns the Factory Limit for the Motor Overtemperature FL exception based on a factory set value related to the Motor Max Winding Temperature of the motor.	°C
681	Optional - PVT	Get		Motor Thermal Overload Factory Limit	REAL	Returns the Factory Limit for the Motor Thermal Overload FL exception based on a factory set value related to the Motor Thermal Overload rating of the motor.	% Motor Rated
682	Optional - PVT	Get		Inverter Overtemperature Factory Limit	REAL	Returns the Factory Limit for the Inverter Overtemperature FL exception.	°C
683	Optional - PVT	Get		Inverter Thermal Overload Factory Limit	REAL	Returns the Factory Limit for the Inverter Thermal Overload FL exception.	% Inverter Rated
684	Optional - PVT	Get		Converter Overtemperature Factory Limit	REAL	Returns the Factory Limit for the Converter Overtemperature FL exception.	°C
685	Optional - PVT	Get		Converter Thermal Overload Factory Limit	REAL	Returns the Factory Limit for the Converter Thermal Overload FL exception.	% Converter Rated
686	Optional - PVT	Get		Bus Regulator Overtemperature Factory Limit	REAL	Returns the Factory Limit for the Bus Regulator Overtemperature FL exception.	°C

Motion Axis Object, Class Code: 42<sub>hex</sub>

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
687	Optional - PVT	Get		Bus Regulator Thermal Overload Factory Limit	REAL	Returns the Factory Limit for the Bus Regulator Thermal Overload FL exception.	% Regulator Rated
688	Optional - PVT	Get		Bus Overvoltage Factory Limit	REAL	Returns the Factory Limit for the Bus Overvoltage FL exception.	Volts
689	Optional - PVT	Get		Bus Undervoltage Factory Limit	REAL	Returns the Factory Limit for the Bus Undervoltage FL exception.	Volts

Table 5-46.52 Exception User Limit Configuration Attributes

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
695	Optional - PVT	Set		Motor Overspeed User Limit	REAL	Sets an incremental speed above the Positive Velocity Limit and below the Negative Velocity Limit that is allowable before throwing a Motor Overspeed UL exception.	Velocity Control Units/Sec
696	Optional - PVT	Set		Motor Overtemperature User Limit	REAL	Sets User Limit for the Motor Overtemperature UL exception.	°C
697	Optional - PVT	Set		Motor Thermal Overload User Limit	REAL	Sets User Limit for the Motor Thermal Overload UL exception.	% Motor Rated
698	Optional - PVT	Set		Inverter Overtemperature User Limit	REAL	Sets User Limit for the Inverter Overtemperature UL exception.	°C
699	Optional - PVT	Set		Inverter Thermal Overload User Limit	REAL	Sets User Limit for the Inverter Thermal Overload UL exception.	% Inverter Rated
700	Optional - PVT	Set		Converter Overtemperature User Limit	REAL	Sets User Limit for the Converter Overtemperature UL exception.	°C
701	Optional - PVT	Set		Converter Thermal Overload User Limit	REAL	Sets User Limit for the Converter Thermal Overload UL exception.	% Converter Rated
702	Optional - PVT	Set		Bus Regulator Overtemperature User Limit	REAL	Sets User Limit for the Bus Regulator Overtemperature UL exception.	°C
703	Optional - PVT	Set		Bus Regulator Thermal Overload User Limit	REAL	Sets User Limit for the Bus Regulator Thermal UL exception.	% Regulator Rated
704	Optional - PVT	Get		Bus Overvoltage User Limit	REAL	Sets User Limit for the Bus Overvoltage UL exception. Unlike the corresponding Factory Limit, which is specified in Volts, the User Limit is based on percent of Nominal Bus Voltage during operation.	% Nominal Bus Voltage

Motion Axis Object, Class Code: 42<sub>hex</sub>

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
705	Optional - PVT	Get		Bus Undervoltage User Limit	REAL	Sets User Limit for the Bus Undervoltage UL exception. Unlike the corresponding Factory Limit, which is specified in Volts, the User Limit is based on percent of Nominal Bus Voltage during operation.	% Nominal Bus Voltage

**5-46.20 Exception Action Configuration Attribute**

The following configuration attribute controls the action performed by the axis object as a result of an exception condition. A unique exception action is defined for each supported exception condition.

**Table 5-46.53 Exception Action Configuration Attribute**

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
667	Required - All	Set		Exception Action	USINT [64]	The Exception Action attribute is 64-element array of enumerated bytes that specifies the action for the associated standard exception. See Semantics for details.	Enumeration: 0 = Ignore (O) 1 = Alarm (O) 2 = Minor Fault (O) 3 = Major Fault (R) 4-255 = reserved
668	Required - All	Set		Exception Action - Mfg	USINT [64]	The Exception Action attribute is 64-element array of enumerated bytes that specifies the action for the associated manufacturer specific exception. See Semantics for details	Enumeration: 0 = Ignore (O) 1 = Alarm (O) 2 = Minor Fault (O) 3 = Major Fault (R) 4-255 = reserved

**5-46.20.1 Semantics****5-46.20.2 Attribute #667-668 – Exception Action**

The Exception Action and Exception Action – Mfg. attributes are 64-element array of enumerated bytes that specifies the action for the associated standard or manufacturer specific exception, respectively. For a given exception, certain actions may not be supported. Each CIP Motion product must specify the available actions for each exception that is supported.

**Table 5-46.54 Drive Status Bit Definitions**

Enum.	Name	Description
0	Ignore	Ignore instructs the axis object to completely ignore the exception condition. For some exceptions that are fundamental to the operation of the device, it may not be possible to Ignore the condition.
1	Alarm	Alarm action simply sets the associated bit in the Alarm Status word but does not otherwise affect axis behavior. For some exceptions that are fundamental to the operation of the axis, it may not be possible to simply indicate the condition as an Alarm.

Motion Axis Object, Class Code: 42<sub>hex</sub>

Enum.	Name	Description
2	Minor Fault	Minor Fault action reports the exception by setting the associated bit in the Fault Status word but does not affect axis behavior. Minor Faults allow the controller to take application specific action in response to the exception condition. For some exceptions that are fundamental to the operation of the axis, it may not be possible to simply indicate the condition as a Minor Fault.
3	Major Fault	<p>Major Fault action results in both setting the associated bit in the Fault Status word and bringing the axis to a stop based on the factory set “best” available stopping method. This “best” stopping method includes both the method of decelerating the motor to a stop and the final state of the axis given the expected level of control still available. The level of control available depends on the specific exception condition and on the configured control mode.</p> <p>The available deceleration methods are defined by the Stopping Mode attribute. Standard stopping modes, listed in decreasing levels of deceleration control, are as follows:</p> <ul style="list-style-type: none"> <li>1. Ramp Decel</li> <li>2. Current Decel</li> <li>3. Coast</li> </ul> <p>In general, the “best” stopping mode is the most controlled deceleration method still available given the exception condition.</p> <p>The final state of the axis in response to the Major Fault exception action can be any one of the following states that are listed in decreasing levels of control functionality:</p> <ul style="list-style-type: none"> <li>1. Hold (Stopped with Holding Torque)</li> <li>2. Disable (Power Structure Disabled)</li> <li>3. Shutdown (DC Bus Power Disabled)</li> </ul> <p>The “best” final state of the axis is the state with the most control functionality still available given the exception condition.</p> <p>If the application requires exception action that is a more severe stopping action than the factory set “best” method, the controller can initiate that action programmatically.</p> <p>If the application requires exception action that is less severe than the factory set “best” method, the controller must configure the axis for a Minor Fault exception action and handle the fault directly. This may put device and motor components at risk and should only be allowed by the device when there is an opportunity, albeit temporal, for the device to remain operational. This is important in applications where the value of the product is higher than the value of the motor or device.</p>
4-255	Reserved	

**5-46.21 Initialization Fault Status Attributes**

The following attribute table contains all initialization fault related attributes associated with a Motion Axis Object instance. Initialization Faults are conditions that can occur during the device initialization process that prevent normal operation of the device.

Motion Axis Object, Class Code: 42<sub>hex</sub>

Table 5-46.55 Initialization Fault Status Attributes

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
669	Required - All	Get		Initialization Fault Status	DWORD	A bit map that represents the state of all standard initialization faults. These faults prevent any motion, and do not have configurable fault actions. Examples of initialization faults are corrupted memory data, calibration errors, firmware startup problems, or an invalid configuration attribute value. Initialization faults cannot be cleared with a Fault Reset service, although a power-cycle provides a new attempt at initialization.	See Semantics
670	Required - All	Get		Initialization Fault Status - Mfg	DWORD	A bit map that represents the state of all manufacturer specific initialization faults. These faults prevent any motion, and do not have configurable fault actions. Examples of initialization faults are corrupted memory data, calibration errors, firmware startup problems, or an invalid configuration attribute value. Initialization faults cannot be cleared with a Fault Reset service, although a power-cycle provides a new attempt at initialization.	See Mfg. Initialization Fault Table (Published in Product Manual)
671	Required - All	Get		Initialization Fault Code	USINT	An 8-bit enumeration that specifies the initialization fault that led to the current faulted state. The value corresponds to the bit position of the standard initialization fault condition, and a value of 0 indicates that no standard initialization faults currently exist. This attribute only reports the first initialization fault that transitioned the axis from an unfaulted state to a faulted state, and then only if that was a standard Initialization Fault. Note that no initialization faults exist if both the Initialization Fault Code and the Mfg. Initialization Fault Code are zero. The Initialization Fault Code and the Mfg. Initialization Fault Code can never both be non-zero at the same time.	See Semantics

Motion Axis Object, Class Code: 42<sub>hex</sub>

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
672	Required - All	Get		Initialization Fault Code - Mfg	USINT	An 8-bit enumeration that specifies the manufacturer specific initialization fault that led to the current faulted state. The value corresponds to the bit position of the initialization fault condition, and a value of 0 indicates that no manufacturer specific initialization faults currently exist. This attribute only reports the first initialization fault that transitioned the axis from an unfaulted state to a faulted state, and then only if that was a Mfg. Initialization Fault. Note that no initialization faults exist if both the Initialization Fault Code and the Mfg. Initialization Fault Code are zero. The Initialization Fault Code and the Mfg. Initialization Fault Code can never both be non-zero at the same time.	See Mfg. Initialization Fault Table (Published in Product Manual)

**5-46.21.1 Semantics****5-46.21.2 Standard Initialization Fault Table**

The following table defines a list of standard faults associated with the Initialization Fault Status attribute and also applicable to the Initialization Fault Code attribute.

**Table 5-46.56 Standard Initialization Fault Table**

Bit	Fault	Bit	Fault
000	Reserved	016	Reserved
001	Boot Block Checksum Fault	017	Reserved
002	Main Block Checksum Fault	018	Reserved
003	Nonvolatile Memory Checksum Fault	019	Reserved
004	Reserved	020	Reserved
005	Reserved	021	Reserved
006	Reserved	022	Reserved
007	Reserved	023	Reserved
008	Reserved	024	Reserved
009	Reserved	025	Reserved
010	Reserved	026	Reserved
011	Reserved	027	Reserved
012	Reserved	028	Reserved
013	Reserved	029	Reserved
014	Reserved	030	Reserved
015	Reserved	031	Reserved

## 5-46.22 Start Inhibit Status Attributes

The following attribute table contains all Start Inhibit related attributes associated with a Motion Axis Object instance. Start Inhibits are conditions that prevent transition of the motion axis from the Stopped State into any of the operational states.

**Table 5-46.57 Start Inhibit Status Attributes**

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
673	Required - PVT	Get		Start Inhibit Status	WORD	A bit map that specifies the current state of all standard conditions that can inhibit starting of the axis:	See Semantics
674	Required - PVT	Get		Start Inhibit Status - Mfg	WORD	A bit map that specifies the current state of all manufacturer specific conditions that can inhibit starting of the axis:	See Mfg. Start Inhibit Table (Published in Product Manual)
675	Required - PVT	Get		Start Inhibit Code	USINT	The Start Inhibit Condition attribute is an enumeration that indicates a condition that presently is causing the axis to be in the start inhibited state. If the inhibiting condition is removed, this attribute will automatically be cleared or else set to another standard inhibiting condition if one is present. If multiple standard start inhibit conditions are present, the Start Inhibit Code is set to the lowest active start inhibit bit number. The Start Inhibit Code and the Mfg. Start Inhibit Code can both be non-zero at the same time.	See Semantics
676	Required - PVT	Get		Mfg Start Inhibit Code	USINT	The Mfg. Start Inhibit Condition attribute is a manufacturer specific enumeration that indicates a condition that presently is causing the axis to be in the start inhibited state. If the inhibiting condition is removed, this attribute will automatically be cleared or else set to another manufacturer specific inhibiting condition if one is present. If multiple manufacturer specific start inhibit conditions are present, the Mfg. Start Inhibit Code is set to the lowest active start inhibit bit number. The Start Inhibit Code and the Mfg. Start Inhibit Code can both be non-zero at the same time.	See Mfg. Start Inhibit Tables (Published in Product Manual)

### 5-46.22.1 Semantics

### 5-46.22.2 Standard Start Inhibit Table

The following table defines a list of standard start inhibits associated with the Start Inhibit Status attribute and also applicable to the Start Inhibit Status Code attribute.

**Table 5-46.58 Standard Start Inhibit Table**

Bit	Start Inhibit	Bit	Start Inhibit
000	Reserved	008	Reserved
001	Enable Input Inactive	009	Reserved
002	Reserved	010	Reserved
003	Reserved	011	Reserved
004	Reserved	012	Reserved
005	Reserved	013	Reserved
006	Reserved	014	Reserved
007	Reserved	015	Reserved

### 5-46.23 Axis Statistical Attributes

The following table includes attributes that provide useful statistics on motion axis operation.

**Table 5-46.59 Axis Statistical Attributes**

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
710	Optional - PVT	Get		Control Power-up Time	REAL	Elapsed time since axis control power was last applied.	Sec
711	Optional - PVT	Get		Cumulative Run Time	REAL	Accumulated time that the axis has been powering the Running state.	Hours
712	Optional - PVT	Get		Cumulative Energy Usage	REAL	Accumulated output energy of the axis.	Kilowatt-Hours
713	Optional – PVT	Get		Cumulative Motor Revs	LINT	Cumulative number of times motor shaft has turned. (Rotary Motors Only)	
714	Optional - PVT	Get		Cumulative Main Power Cycles	DINT	Cumulative number of times AC Mains has been cycled.	
715	Optional - PVT	Get		Cumulative Control Power Cycles	DINT	Cumulative number of times Control Power has been cycled.	

### 5-46.24 Axis Info Attributes

The following table includes attributes that provide information about the associated hardware capabilities of Motion Axis Object instance.

**Table 5-46.60 Axis Info Attributes**

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
720	Required - PVT	Get		Inverter Rated Output Voltage	REAL	Drive inverter output voltage rating. This value is hard coded in the device.	Volts (RMS)
721	Required - PVT	Get		Inverter Rated Output Current	REAL	Drive inverter output current rating. This value is hard coded in the device.	Amps (RMS)

**Motion Axis Object, Class Code: 42<sub>hex</sub>**

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
722	Required - PVT	Get		Inverter Rated Output Power	REAL	Drive inverter output power rating. This value is hard coded in the device.	Kilowatts
723	Optional - PVT	Get		Converter Rated Output Current	REAL	Drive converter output current rating. This value is determined by the motion axis from the associated converter.	Amps
724	Optional - PVT	Get		Converter Rated Output Power	REAL	Drive converter output power rating. This value is determined by the motion axis from the associated converter.	Kilowatts

**5-46.25 General Purpose I/O Attributes**

The following table includes attributes that provide to general purpose analog and digital I/O associated with the Motion Axis Object instance.

**Table 5-46.61 Drive General Purpose I/O Attributes**

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
730	Optional - PVT	Get		Digital Inputs	DWORD	A 32-bit word with whose bits can be assigned by the vendor to general purpose digital inputs.	<b>Vendor Specific Bit Map</b>
731	Optional - PVT	Set		Digital Outputs	DWORD	A 32-bit word with whose bits can be assigned by the vendor to general purpose digital outputs.	Vendor Specific Bit Map
732	Optional - PVT	Get		Analog Input 1	REAL	General purpose analog input 1 level.	% Full Scale
733	Optional - PVT	Get		Analog Input 2	REAL	General purpose analog input 2 level.	% Full Scale
734	Optional - PVT	Set		Analog Output 1	REAL	General purpose analog output 1 level.	% Full Scale
735	Optional - PVT	Set		Analog Output 2	REAL	General purpose analog output 2 level.	% Full Scale

**5-46.26 Local Mode Attributes**

The following attributes tables contain all local mode attributes associated with a Motion Axis Object instance. These attributes govern the behavior of the Motion Axis Object instance when a local (i.e. non-CIP Motion) device interface requests control of axis behavior.

Motion Axis Object, Class Code: 42<sub>hex</sub>

Table 5-46.62 Local Mode Configuration Attributes

Attr ID	Need in Implem	Access Rule	N V	Attribute Name	Data Type	Description of Attribute	Semantics of Values
750	Optional	Set		Local Control	USINT	Mechanism for controller to allow a local interface to request local control. Control implies in this case the ability to change the state or behavior of the axis. Local Control Not Allowed configures the device to prevent a local interface from taking control of the axis. Local Control Conditionally Allowed configures the device to prevent a local interface from taking control of the axis while the axis is in an Operational State, such as Running or Testing. Local Control Allowed configures the device to allow a local interface to take control of the axis, even in an Operational State. Some devices may not support this state.	Enumeration: 0 = Local Control Not Allowed 1 = Local Control Conditionally Allowed 2 = Local Control Allowed (O)

**5-46.27 Common Services**

The Motion Axis Object provides the following CIP Common Services.

Table 5-46.63 Motion Axis Object – CIP Common Services

Service Code (Hex)	Need in Implementation		Service Name	Description of Service
	Class	Instance		
03 <sub>hex</sub>	Required	Required	Get_Attributes_List	Returns the contents of the selected gettable attributes for given instance.
04 <sub>hex</sub>	Required	Required	Set_Attributes_List	Sets the content of the selected settable attributes for given instance.
0E <sub>hex</sub>	Required	Required	Get_Attributes_Single	Returns the contents of the selected gettable attribute for given instance.
10 <sub>hex</sub>	Required	Required	Set_Attributes_Single	Sets the content of the selected settable attribute for given instance.
1C <sub>hex</sub>	Optional	N/A	GroupSync	Passes controller's System Time Offset to drive and reports if this drive can be synchronized with the controller as part of a group of associated drives. If the drive can be synchronized, the drive also returns the drive's System Time Offset to the controller in the response.

## 5-46.28 Service Specific Data

Common Services supported by the Motion Axis Object have associated Request Data and Response Data. The format and syntax of the Service Specific Data depends on the specified Service Code. Descriptions of the Service Specific Data for the Set\_Attribute\_List, Get\_Attribute\_List, Set\_Attribute\_Single and Get\_Attribute\_Single services are already defined in Appendix A. A description of the Service Specific Data associated with the GroupSync service added as part of the CIP Sync extensions to CIP is detailed in the following section.

### 5-46.28.1 GroupSync Service

The GroupSync service is used to synchronize the Motion Axis Object associated with this drive node to the controller issuing the service request and, ultimately, to all other drives comprising the controller's Motion Group. The service request passes the controller's System Time Offset and replies with a 1 if the drive is presently group synchronized to the IEEE-1588 time master or a 0 if it is not group synchronized.

**Table 5-46.64 GroupSync Service Request Data Structure**

Name	Type	Description of Request Parameter	Semantics of Values
System Time Offset	LINT	Specifies the current System Time Offset of the controller:	Nanoseconds (CIP Sync Absolute)

The System Time Offset value is used to initialize the last Controller Time Offset value used as part of the System Time Offset Check and Compensation algorithm.

**Table 5-46.65 GroupSync Service Response Data Structure**

Name	Type	Description of Response Parameter	Semantics of Values
Test Status	BOOL	Boolean value that indicates whether or not the drive is presently group synchronized.	0 = drive not synchronized 1 = drive synchronized
System Time Offset	LINT	Specifies the current System Time Offset of the drive:	Nanoseconds (CIP Sync Absolute)

There are two class attributes used by the Motion Axis Object to determine if the drive is group synchronized.

1. Sync Update Delay
2. Sync Threshold

The Sync Update Delay attribute determines the minimum time to wait after the last System Time Offset change greater than Sync Threshold before the device is considered to be group synchronized. This value is the product of the number of networks hops the device is away from the grandmaster, as determined by the Time Sync object attribute, Steps Removed, and the time synchronization interval, as determined by the Time Sync object attribute, Sync Interval. The Sync Update Delay attribute is typically several seconds.

The Sync Threshold is the value used to determine if the device's local clock is frequency synchronized to the master System Time clock. System Time Offset changes at the motion device that exceed the Sync Threshold are considered adjustments to the System Time reference system rather than minor adjustment due to local clock drift. This kind of adjustment is indicative of a motion device whose local clock is not yet fully synchronized to the master clock. Though application dependent, the minimum Sync Threshold value depends on the motion device's time sync implementation, but is typically on the order of microseconds for a hardware assisted clock implementation.

**Motion Axis Object, Class Code: 42<sub>hex</sub>**

Once the controller and all the devices in the Motion Group are group synchronized, the controller can then set the Synchronous Control bit in the Controller-to-Device Connection header of the next cyclic packet sent to the motion devices indicating that the device can schedule its next Device-to-Controller Connection update based on the Controller Time Stamp and Controller Update Period contained in the connection data. The motion device indicates that it is now sending scheduled connection data based on the Controller Update Period by setting the Sync Mode bit in the Device-to-Controller Connection header of the next cyclic packet sent to the controller. Data exchange between the controller and the device proceed thereafter according the CIP Motion Timing Model.

**5-46.29 Object Specific Services**

The Motion Axis Object provides the following Object Specific Services.

**Table 5-46.66 Motion Axis Object – Object Specific Services**

Service Code	Need in Implementation		Service Name	Description of Service
	Class	Instance		
4B <sub>hex</sub>		Required	Get Axis Attributes List	Returns the contents of the selected gettable Motion Axis Object attributes.
4C <sub>hex</sub>		Required	Set Axis Attributes List	Sets the content of the selected settable Motion Axis Object attributes.
4D <sub>hex</sub>		Required	Set Cyclic Write List	Sets the attribute contents of the Cyclic Write Data block.
4E <sub>hex</sub>		Required	Set Cyclic Read List	Sets the attribute contents of the Cyclic Read Data block.
4F <sub>hex</sub>		Optional - FC	Run Motor Test	Initiates the selected test on the motor to measure various motor parameters.
50 <sub>hex</sub>		Optional - FC	Get Motor Test Data	Returns the results of the preceding Run Motor Test service.
51 <sub>hex</sub>		Required - C	Run Inertia Test	Initiates the selected test on the motor to measure the inertia.
52 <sub>hex</sub>		Required - C	Get Inertia Test Data	Returns the results of the preceding Run Inertia Test service.
53 <sub>hex</sub>		Required	Run Hookup Test	Initiates the selected test to determine the condition of the motor and feedback device connections.
54 <sub>hex</sub>		Required	Get Hookup Test Data	Returns the results of the preceding Run Hookup Test service.
56-63 <sub>hex</sub>			Reserved	

**5-46.30 Service Specific Data**

Object Specific Services supported by the Motion Axis Object have associated Request Data and Response Data. The format and syntax of the Service Specific Data depends on the specified Service Code. All parameters contained in the Service Specific Data are word aligned. This is true regardless of whether the service specific request data is passed in the Controller-to-Device Connection or as part of an Explicit messaging connection. A description of the Service Specific Data associated with each service is shown below.

### 5-46.30.1 Get Axis Attribute List

The Get Axis Attribute List service provides a mechanism to read the current value of one or more Motion Axis Object attributes, including attributes having a multi-dimensional array data type. The buffer/array transfer mechanism can be used to transfer large data log arrays from the device to facilitate features like high speed trending.

The format of the Request Data Block for this service is shown below.

**Figure 5-46.16 Get Axis Attribute List Request Format**

Get Axis Attribute List Request Format		
Number of Attributes	-	-
Attr ID 1	Attr 1 Dimension	Attr 1 Element Size
Attr Start Index 1 (array only)	Attr Data Elements 1 (array only)	
...	...	
Attr ID 2	Attr 2 Dimension	Attr 2 Element Size
Attr Start Index 2 (array only)	Attr Data Elements 2 (array only)	
...	...	
Attr ID n	Attr n Dimension	Attr n Element Size
Attr Start Index n (array only)	Attr Data Elements n (array only)	
...	...	

Definitions of the individual parameters in this data structure are as follows:

- **Number of Attributes** – represents the number of attributes contained in the Get Axis Attribute service request.
- **Attr ID** – identifies the targeted Motion Axis Object attribute to get.
- **Attr Dimension** – determines the dimension of the attribute array. A dimension of zero means the attribute is a singular data element and, therefore, not really an array at all. Multidimensional arrays (dimension > 1) are supported by adding additional Attr Start Index and Attr Data Elements values. If there is an error associated with a specific requested get attribute operation the device shall indicate this by setting the Attr Dimension to 0xFF, in which case the Element Size field contains the specific error code. When an error code is present neither the array index parameters nor the attribute data fields are returned.
- **Attr Element Size** – determines the size, in bytes, of the attribute data element. Data elements must be word aligned; 32-bit words are 32-bit aligned and 16-bit words are 16-bit aligned. Padding may be added to maintain word alignment. If there is an error associated with a specific requested get attribute operation the Element Size field contains the CIP General Status error code. When an error code is present neither the array index parameters nor the attribute data fields are returned.
- **Attr Start Index** – identifies the starting index for the array of attribute values in the Attr Data section. This field is only present when the attribute data type is an array (i.e., dimension > 0).
- **Attr Data Elements** – determines the number of data element values in the Attr Data section for the associated index. This field is only present when the attribute data type is an array (i.e., dimension > 0).

The structure of Response Data Block for this service is as follows:

**Figure 5-46.17 Get Axis Attribute List Response Format**

Get Axis Attribute List Response Format		
Number of Attributes	-	
Attr ID 1	Attr 1 Dimension	Attr 1 Element Size
Attr Start Index 1 (array only)	Attr Data Elements 1 (array only)	
...	...	
	Attr Data 1	
	...	
Attr ID 2	Attr 2 Dimension	Attr 2 Element Size
Attr Start Index 2 (array only)	Attr Data Elements 2 (array only)	
...	...	
	Attr Data 2	
	...	
Attr ID n	Attr n Dimension	Attr n Element Size
Attr Start Index n (array only)	Attr Data Elements n (array only)	
...	...	
	Attr Data n	
	...	

Definitions of the individual parameters in this data structure are identical to the request data structure with the addition of the Attribute Data element, which is defined as follows:

- **Attr Data** – contains the current value(s) of the targeted Motion Axis Object attribute indicated by the Attr ID. If the attribute is an array, the value(s) are listed according to the Attr Start Index and the number of Attr Data Elements. For multidimensional arrays (dimension > 1), the sequence of data move sequentially through the indices from right to left. Here are some examples,

Figure 5-46.18 illustrates a Get\_Attribute\_List Request for a simple 4-byte DINT scalar attribute. Note that since the Attr Dim is 0, the Attr Start Index and Attr Data Element fields are omitted.

**Figure 5-46.18 — Get Axis Attribute List Response – Example 1**

Get Axis Attribute List Response – 0 Dimensional Data Example		
Number of Attributes = 1	-	
Attr ID = 25	Attr Dim = 0	Attr Elem Size = 4
	Attr Data = ?	

Figure 5-46.19 illustrates a Get\_Attribute\_List Request for the first three elements of a one-dimensional array attribute that is a UINT. Since the Attr Elem Size for this attribute is a 2-byte value, a Pad byte is added to maintain word alignment for any connection data to follow.

**Motion Axis Object, Class Code: 42<sub>hex</sub>****Figure 5-46.19 — Get Axis Attribute List Response – Example 2**

Get Axis Attribute List Response – 1 Dimensional Array Example		
Number of Attributes = 1	-	-
Attr ID = 43	Attr Dim = 1	Attr Elem Size = 2
Attr Start Index 1 = 0	Attr Data Elements 1 = 3	
Attr Data (0) = ?	Attr Data (1) = ?	
Attr Data (2) = ?	(Pad)	

Figure 5-46.20 illustrates a Get\_Attribute\_List Response for six elements of a two-dimensional array of REALs. Note that the Attr Data sequences through the left most array index first beginning with the Attr Start Index 1 of 0.

**Figure 5-46.20 Get Axis Attribute List Response – Example 3**

Get Axis Attribute List Response – 2 Dimensional Array Example		
Number of Attributes = 1	-	-
Attr ID = 27	Attr Dim = 2	Attr Elem Size = 4
Attr Start Index 1 = 0	Attr Data Elements 1 = 2	
Attr Start Index 2 = 2	Attr Data Elements 2 = 3	
Attr Data (0, 2) = ?		
Attr Data (0, 3) = ?		
Attr Data (0, 4) = ?		
Attr Data (1, 2) = ?		
Attr Data (1, 3) = ?		
Attr Data (1, 4) = ?		

If there is an error associated with a specific requested get attribute operation the device shall indicate this by setting the Attr Dimension to 0xFF, in which case the Element Size field may contain the specific error code. When an error code is present neither the array index parameters nor the attribute data fields are returned. Figure 5-46.21 below illustrates such a case, where Attribute 29 is not supported by the targeted motion axis as indicated by the CIP General Status Code of 0x14.

**Figure 5-46.21 Get Axis Attribute List Response – Example 4**

Get Axis Attribute List Response – Attribute Error Example		
Number of Attributes = 3	-	-
Attr ID = 25	Attr Dim = 0	Attr Elem Size = 4
Attr Data = 43.5		
Attr ID = 29	Attr Dim = 0xFF	Error Code = 0x14
Attr ID = 43	Attr Dim = 1	Attr Elem Size = 2
Attr Start Index 1 = 0	Attr Data Elements 1 = 3	
Attr Data [0] = 55	Attr Data [1] = 45	
Attr Data [2] = 35	(Pad)	

### 5-46.30.2 Set Axis Attribute List

The Set Axis Attribute List service provides a mechanism to write a value to one or more settable Motion Axis Object attributes, including attributes having a multi-dimensional array data type. The buffer/array transfer mechanism can be used to build parameter tables in the drive for indexing, or camming applications.

The format of the Request Data Block for this service is shown below.

**Figure 5-46.22 Set Axis Attribute List Request Format**

← 32-bit Word →		
<b>Set Drive Attribute List Request Format</b>		
Number of Attributes	-	
Attr ID 1	Attr 1 Dimension	Attr 1 Element Size
Attr Start Index 1 (array only)	Attr Data Elements 1 (array only)	
...	...	
	Attr Data 1	
	...	
Attr ID 2	Attr 2 Dimension	Attr 2 Element Size
Attr Start Index 2 (array only)	Attr Data Elements 2 (array only)	
...	...	
	Attr Data 2	
	...	
Attr ID 3	Attr n Dimension	Attr n Element Size
Attr Start Index n (array only)	Attr Data Elements n (array only)	
...	...	
	Attr Data 3	
	...	

Definitions of the individual parameters in this data structure are as follows:

- **Number of Attributes** – represents the number of attributes contained in the Set Axis Attribute service request.
- **Attr ID** – identifies the targeted Motion Axis Object configuration attribute to set.
- **Attr Dimension** – determines the dimension of the attribute array. A dimension of zero means the attribute is a singular data element and, therefore, not really an array at all. Multidimensional arrays (dimension > 1) are supported by adding additional Attr Start Index and Attr Data Elements values prior to the Attr Data sequence.
- **Attr Element Size** – determines the size, in bytes, of the attribute data element. Data elements must be word aligned; 32-bit words are 32-bit aligned and 16-bit words are 16-bit aligned. Padding may be added to maintain word alignment.
- **Attr Start Index** – identifies the starting index for the array of attribute values in the Attr Data section. This field is only present when the attribute data type is an array (i.e., dimension > 0).
- **Attr Data Elements** – determines the number of data element values based on the start index that included in the Attr Data section. This field is only present when the attribute data type is an array (i.e., dimension > 0).

**Motion Axis Object, Class Code: 42<sub>hex</sub>**

- **Attr Data** – contains the new value(s) that are to be applied to the targeted drive configuration attribute indicated by the Attr ID. If the attribute is an array, the new value(s) are applied according to the Attr Start Index and the number of Attr Data Elements. For multidimensional arrays (dimension > 1), the list of Attr Data elements moves sequentially through the indices from left to right. The data type of the Attr Data must match the data type of the attribute specified by the Attr ID.

Figure 5-46.23 illustrates a Set\_Attribute\_List Request for a simple 4-byte scalar attribute. Note that since the Attr Dim is 0, the Attr Start Index and Attr Data Element fields are omitted.

**Figure 5-46.23 Set Axis Attribute List Request – Example 1**

Set Axis Attribute List Request – 0 Dimensional Data Example		
Number of Attributes = 1	-	
Attr ID = 25	Attr Dim = 0	Attr Elem Size = 4
Attr Data = ?		

Figure 5-46.24 illustrates a Set\_Attribute\_List Request for the first three elements of a one-dimensional array of UINTs. Since the Attr Elem Size is 2-bytes, a Pad byte is added to maintain word alignment for any connection data to follow.

**Figure 5-46.24 Set Axis Attribute List Request – Example 2**

Set Axis Attribute List Request – 1 Dimensional Array Example		
Number of Attributes = 1	-	
Attr ID = 43	Attr Dim = 1	Attr Elem Size = 2
Attr Start Index 1 = 0	Attr Data Elements 1 = 3	
Attr Data (0) = ?	Attr Data (1) = ?	
Attr Data (2) = ?	(Pad)	

Figure 5-46.25 illustrates a Set\_Attribute\_List Request for six elements of a two-dimensional array of REALs. Note that the Attr Data sequences through the left most array index first beginning with the Attr Start Index 1 of 0.

**Figure 5-46.25 Set Axis Attribute List Request – Example 3**

Set Axis Attribute List Request – 2 Dimensional Array Example		
Number of Attributes = 1	-	
Attr ID = 27	Attr Dim = 2	Attr Elem Size = 4
Attr Start Index 1 = 0	Attr Data Elements 1 = 2	
Attr Start Index 2 = 2	Attr Data Elements 2 = 3	
Attr Data (0, 2) = ?		
Attr Data (0, 3) = ?		
Attr Data (0, 4) = ?		
Attr Data (1, 2) = ?		
Attr Data (1, 3) = ?		
Attr Data (1, 4) = ?		

The structure of Response Data Block for this service is as follows:

**Figure 5-46.26 Set Axis Attribute List Response Format**

← 32-bit Word →		
Set Axis Attribute List Response Format		
Number of Attributes	-	-
Attr ID 1	Attr Status 1	-
Attr ID 2	Attr Status 2	-
...		
Attr ID n	Attr Status n	-

- **Attr ID** – identifies the targeted Motion Axis Object configuration attribute of the set list request.
- **Attr Status** – indicates whether the set list action for the targeted attribute was successful. An Attribute Status value is zero if the targeted attribute was successfully written. A non-zero Attr Status value indicates that an error occurred that prevented the attribute from being updated. The error codes follow the CIP General Status Codes.

#### 5-46.30.3 Set Cyclic Write List

The Set Cyclic Write List service provides a mechanism to determine the list of attributes to be passed as part of the Cyclic Write Data Block of the Controller-to-Device connection. The format of the Request Data Block for this service is shown below.

**Figure 5-46.27 Set Cyclic Write List Request Format**

← 32-bit Word →		
Set Cyclic Write List Request Format		
Number of Attributes	-	-
Attr ID 1	Attr ID 2	-
...		

**Number of Attributes** – represents the number of attributes contained in the Cyclic Write List.

**Attr ID** – identifies the specific drive configuration attribute that is to be updated via the Cyclic Write Data Block of the Controller-to-Device connection. The Attr ID determines the data type and implied semantics of the data based on the specifications for the associated attribute.

The ordering of the attribute data in the Cyclic Write Data Block is determined by the ordering of the Attr IDs in the Set Cyclic Write List request. Attribute data elements must be word aligned; 32-bit words are 32-bit aligned and 16-bit words are 16-bit aligned. Padding may be added to maintain word alignment.

The structure of Response Data Block for this service is as follows:

**Figure 5-46.28 Set Cyclic Write List Response Format**

← 32-bit Word →

Set Cyclic Write List Response Format		
Number of Attributes	Cyclic Write Block ID	
Attr ID 1	Attr Status 1	-
Attr ID 2	Attr Status 2	-
...		
Attr ID n	Attr Status n	-

**Number of Attributes** – represents the number of attributes contained in the requested Cyclic Write List.

**Cyclic Write Block ID** - If all attributes in the requested list can be supported in the Set Cyclic Write Data Block, the Set Cyclic Write List response provides a new Cyclic Write Block ID that is simply the increment of the current Cyclic Write Block ID. This new Cyclic Write Block ID can be used in the next Controller-to-Device Connection update. If the Set Cyclic Write List is not successful, the Cyclic Write Block ID remains the current value.

**Attr ID** – identifies the specific drive configuration attribute that was requested to be updated via the Cyclic Write Data Block of the Controller-to-Device Connection.

**Attr Status** – The Attr Status value is zero if the associated attribute indicated by the Attr ID can be supported in the Set Cyclic Write Data Block. A non-zero Attr Status code indicates the specific reason why the associated attribute cannot be supported. These codes follow the CIP General Status Codes. If the Attr Status of one or more Attribute IDs in the list indicates an error, the Set Cyclic Write List request is considered unsuccessful.

#### 5-46.30.4 Set Cyclic Read List

The Set Cyclic Read List service provides a mechanism to determine the list of attributes to be passed as part of the Cyclic Read Data Block of the Device-to-Controller connection. The format of the Request Data Block for this service is shown below.

**Figure 5-46.29 Set Cyclic Read List Request Format**

← 32-bit Word →

Set Cyclic Read List Request Format		
Number of Attributes		-
Attr ID 1		Attr ID 2
...		

**Number of Attributes** – represents the number of attributes contained in the Cyclic Read List.

**Attr ID** – identifies the specific drive status or signal attribute that is to be updated via the Cyclic Read Data Block of the Device-to-Controller connection. The Attr ID determines the data type and implied semantics of the data based on the specifications for the associated attribute.

The ordering of the attribute data in the Cyclic Read Data Block is determined by the ordering of the Attr IDs in the Set Cyclic Read List request. Attribute data elements must be word aligned; 32-bit words are 32-bit aligned and 16-bit words are 16-bit aligned. Padding may be added to maintain word alignment.

The structure of Response Data Block for this service is as follows:

**Figure 5-46.30 Set Cyclic Read List Response Format**

← 32-bit Word →

Set Cyclic Read List Response Format		
Number of Attributes	Cyclic Read Block ID	
Attr ID 1	Attr Status 1	-
Attr ID 2	Attr Status 2	-
...		
Attr ID n	Attr Status n	-

**Number of Attributes** – represents the number of attributes contained in the requested Cyclic Read List.

**Cyclic Read Block ID** - If all attributes in the requested list can be supported in the Set Cyclic Read Data Block, the Set Cyclic Read List response provides a new Cyclic Read Block ID that is simply the increment of the current Cyclic Read Block ID. This new Cyclic Read Block ID can be used in the next Device-to-Controller Connection update. If the Set Cyclic read List is not successful, the Cyclic Read Block ID remains the current value.

**Attr ID** – identifies the specific drive configuration attribute that was requested to be updated via the Cyclic Read Data Block of the Device-to-Controller connection.

**Attr Status** – The Attr Status value is zero if the associated attribute indicated by the Attr ID can be supported in the Set Cyclic Read Data Block. A non-zero Attr Status code indicates the specific reason why the associated attribute cannot be supported. These codes follow the CIP General Status Codes. If the Attr Status of one or more Attribute IDs in the list indicates an error, the Set Cyclic Read List request is considered unsuccessful.

#### 5-46.30.5 Run Motor Test

The Run Motor Test service initiates various test operations motor to determine important motor parameters. The Static Test applies a directionally static flux to the motor, while the Dynamic Test performs dynamic tests on the motor that produce motion. The parameters measured by these tests are retrieved by the controller via the Get Motor Test Data command.

**Table 5-46.67 Run Motor Test Request Data Structure**

Name	Type	Description of Request Parameter	Semantics of Values
Test Direction	USINT	Enumeration that selects test direction:	Enumeration: 0 = forward 1 = reverse 2 = forward bi-directional 3 = reverse bi-direction
Test Type	USINT	Enumeration that specifies the type of test to perform:	Enumeration: 0 = static test 1 = dynamic test

There are no specific response parameters required for this service.

#### 5-46.30.6 Get Motor Test Data

The Get Motor Test Data service provides access to the motor parameter measurements generated by the last Motor Test command. The controller uses this data to update the device with the best estimates of critical motor parameters.

**Motion Axis Object, Class Code: 42<sub>hex</sub>**

There are no specific request parameters required for this service.

**Table 5-46.68 Get Motor Test Response Data Structure**

Name	Type	Description of Response Parameter	Semantics of Values
Test Status	USINT	Indicates the result of the last Motor Test command:	Enumeration: 0 = test process successful 1 = test in progress 2 = test process aborted 3 = test process timed-out 4 = test process faulted
Stator Resistance	REAL	Measured phase-to-phase stator resistance of the motor.	Ohms
Leakage Inductance	REAL	Measured phase-to-phase leakage inductance of the motor.	Henries
Flux Current	REAL	Measured current to achieve full induction motor flux. (IM only)	Amps
Slip Frequency	REAL	Measured slip at rated induction motor current. (IM only)	Hertz

**5-46.30.7 Run Inertia Test**

The Run Inertia Test service performs an acceleration and deceleration ramp on the axis and makes timing measurements in the process. The timing measurements are accessed by the controller via the Get Inertia Test Data command and ultimately used to calculate an accurate inertia value for the motor and load.

**Table 5-46.69 Run Inertia Test Request Data Structure**

Name	Type	Description of Request Parameter	Semantics of Values
Test Direction	USINT	Enumeration that selects test direction:	Enumeration: 0 = forward 1 = reverse
Test Velocity	REAL	Determines the maximum velocity that will be reached during the test profile.	Velocity Feedback Counts / Sec
Test Torque	REAL	Determines the maximum torque that is applied during the test profile.	% Motor Rated
Test Travel Limit	REAL	Establishes the maximum excursion of the axis in Feedback Counts allowed in the test direction.	Feedback Counts

There are no specific response parameters required for this service.

**5-46.30.8 Get Inertia Test Data**

The Get Inertia Test Data service provides access to the timing measurements generated by the last Run Inertia Test service. The controller uses this data to ultimately calculate an accurate inertia value for the motor and load.

There are no specific request parameters required for this service.

**Motion Axis Object, Class Code: 42<sub>hex</sub>****Table 5-46.70 Get Inertia Test Response Data Structure**

Name	Type	Description of Response Parameter	Semantics of Values
Test Status	USINT	Enumeration that indicates the result of the last Run Inertia Test service:	Enumeration: 0 = test process successful 1 = test in progress 2 = test process aborted 3 = test process timed-out 4 = test process fault 5 = test reached limit 6 = test polarity incorrect 7 = test speed set too low
Accel Time	REAL	Measured acceleration time to reach test velocity.	Sec
Decel Time	REAL	Measured deceleration time to reach zero speed from test velocity.	Sec

**5-46.30.9 Run Hookup Test**

The Run Hookup Test service performs a number of different tests to check for proper interface to the motor and/or feedback device. Three tests are currently supported. The Motor/Feedback test is an active test that attempts to move the axis a specified distance given by the Test Increment. The Feedback test monitors the axis while an external agent moves it and indicates success if the axis feedback count exceeds the distance specified by the Test Increment. Finally, the Commutation Test, which applies only to PM motors, applies current to the motor to align the rotor, check for proper phasing of a Hall sensor if applicable, and measure the Commutation Offset. The resultant data generated by these test can be accessed by the Get Test Data service and used by the controller to automatically set the proper polarities of the feedback interface and the motor interface.

**Table 5-46.71 Run Hookup Test Request Data Structure**

Name	Type	Description of Request Parameter	Semantics of Values
Test Increment	REAL	Establishes the distance that the axis needs to travel, in Feedback Counts to indicate a successful test.	Feedback Counts
Hookup Test	USINT	Determines the specific test to run:	Enumeration: 0 = motor/feedback test 1 = feedback test 2 = commutation test

There are no specific response parameters required for this service request.

**5-46.30.10 Get Hookup Test Data**

The Get Hookup Test Data service provides access to the hookup test results generated by the last Run Hookup Test service. The controller uses this data to flag a wiring problem to the user and to calculate polarity configuration bit parameters for the axis.

There are no specific request parameters required for this service.

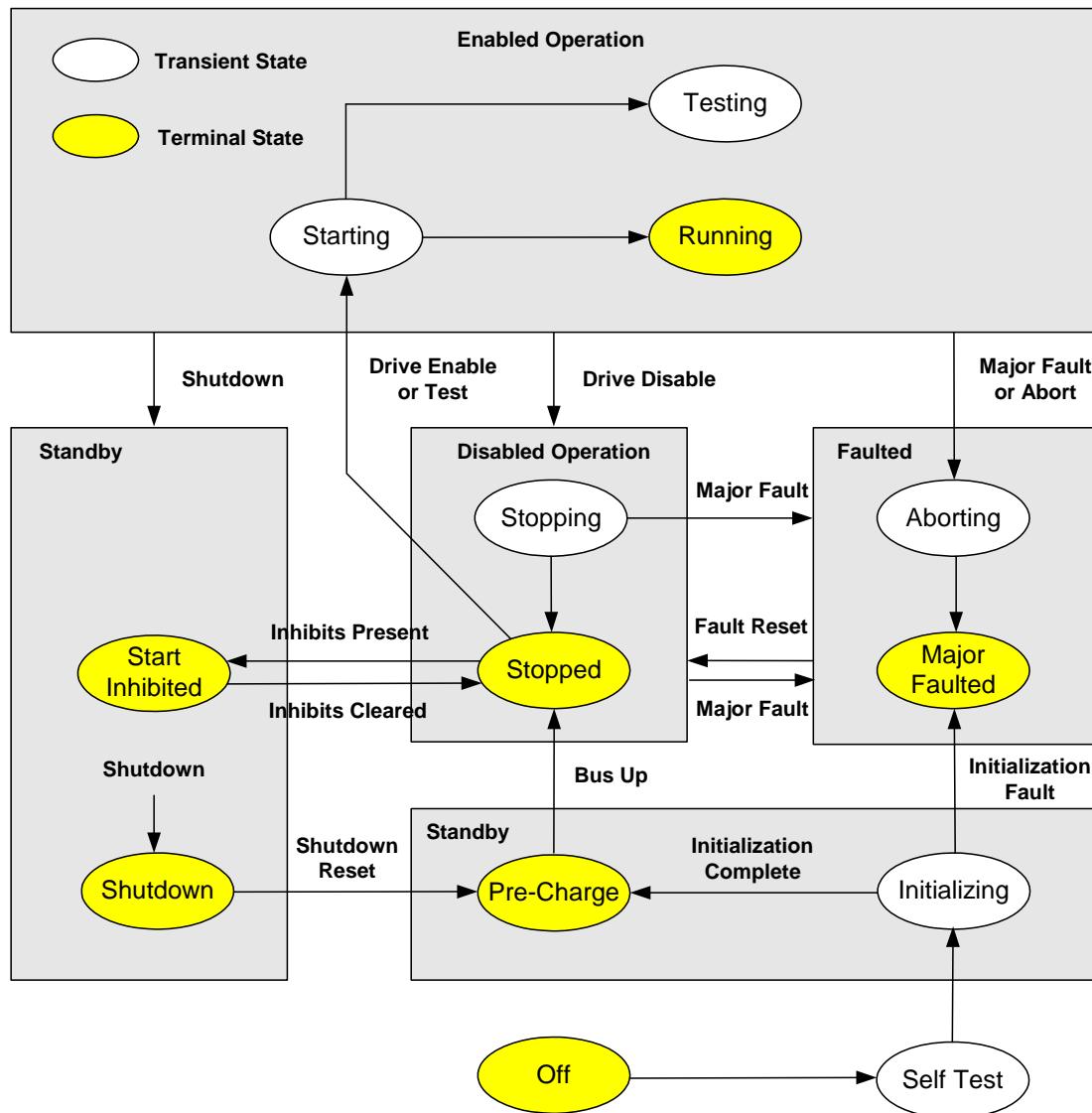
**Motion Axis Object, Class Code: 42<sub>hex</sub>****Table 5-46.72 Get Hook-up Test Response Data Structure**

Name	Type	Description of Response Parameter	Semantics of Values
Test Status	USINT	Enumeration that indicates the result of the last Hookup Test command:	Enumeration: 0 = test process successful 1 = test in progress 2 = test process aborted 3 = test process timed-out 4 = test process faulted
Test Direction Forward	USINT	Enumeration that indicates the direction of axis motion associated with the last Hookup Test command.	Enumeration: 0 = forward 1 = reverse
Hall Commutation Map	USINT	When performing a Commutation Test on a PM motor feedback device the test checks if the UVW phases of the Hall Sensor match the UVW phases of the Motor. If it is determined that the UVW phases for the motor and hall device do not match, typically due to incorrect wiring, this parameter reports the necessary commutation map can be used to compensate the mismatch. The enumerated choices listed assume the motor is wired correctly as UVW and describes different mappings of the UVW Hall sensor inputs that if applied to the device would correct the phasing.	Enumeration: 0 = UVW (normal) 1 = UWV 2 = WVU 3 = WUV 4 = VUW 5 = VWU
Commutation Offset	REAL	Offset measured during the Commutation Test that must be applied to the motor position accumulator in order to align the Electrical Angle signal with motor stator's orientation	Electrical Degrees

**5-46.31 Behavior****5-46.31.1 State Model**

The Motion Axis Object State Model is based on elements of the S88 and Pack/ML standard state models. The current state of the Motion Axis Object instance is indicated by the Axis State attribute. State transitions can be initiated either directly via the Axis Control request mechanism or by conditions that occur during device operation. The diagram below shows the basic operating states of the Motion Axis Object.

Figure 5-46.31 Motion Axis Object State Model



### 5-46.31.2 State Control

The primary method for changing the state of a Motion Axis Object instance is via the Axis Control/Axis Response mechanism that is built into the CIP Motion I/O Connection header. Changing the state of the motion axis is simply performed by placing the appropriate Request Code in the Axis Control element of the Controller-to-Device Connection header. The Control Request codes currently defined in this object are as follows:

**Table 5-46.73 Axis Control Request Code**

Request Code	Requested Operation
0	No Request
1	Enable Request
2	Disable Request
3	Shutdown Request
4	Shutdown Reset Request
5	Abort Request
6	Fault Reset Request
7-127	Reserved
128-255	Vendor Specific

When a state transition is requested via Axis Control element of the Controller-to-Device connection, the axis initiates the state transition and then acknowledges the transition request via the Axis Response attribute when the requested state transition completes or is determined unsuccessful. The Acknowledge Codes for the Axis Response element are the same as the Request Codes for the Axis Control element.

**Table 5-46.74 Axis Response Acknowledge Codes**

Acknowledge Code	Axis Response
0	No Acknowledge
1	Enable Acknowledge
2	Disable Acknowledge
3	Shutdown Acknowledge
4	Shutdown Reset Acknowledge
5	Abort Acknowledge
6	Fault Reset Acknowledge
7-127	Reserved
128-255	Vendor Specific

The criteria for successful completion of the requested operation depend on the specific Request Code as shown in the table below.

**Table 5-46.75 Completion Criteria for Requested Operation**

Request Code	Completion Criteria
1	Axis successfully transitions to Starting state.
2	Axis successfully transitions to Stopped state.
3	Axis successfully transitions to Shutdown state
4	Axis successfully transitions to Stopped or Inhibited state
5	Axis successfully transitions to Faulted state.
6	Axis successfully transitions to Stopped, Inhibited, or Shutdown state.
7-127	Reserved
128-255	Vendor Specific

**Motion Axis Object, Class Code: 42<sub>hex</sub>**

When the controller receives the Acknowledge Code, it then zeroes the matching Request Code. When the drive sees the Request Code zeroed, it clears the Acknowledge Code and the associated Response Status.

If the axis cannot transition to the requested state, the specific error condition is indicated by a non-zero Response Status attribute value that is passed along with the Axis Response at part of the Device-to-Controller connection.

In the case of a Fault Reset, if any fault generating exception conditions still exist, an Error Code is generated to indicate that the axis cannot transition out of the Faulted State until the associated exception conditions have been cleared.

This behavior is illustrated in the tables below.

**Table 5-46.76 Successful Axis Control Request Cycle**

<b>Controller Action</b>	<b>Connection Data</b>	<b>Drive Action</b>
Controller sets Request Code in the Axis Control element to request a state change.	→ Drive Control = x	Drive sees non-zero Request Code, x, in Axis Control element and initiates the requested state change.
	← Axis Response = x ← Response Status = 0	If state change operation is successful, the device acknowledges the state change by setting the Acknowledge Code in the Axis Response element to the originating Request Code, x. The Response Status element is set to 0 indicating success.
Controller sees the non-zero Acknowledge Code in Axis Response element and zeroes the originating Request Code in the Axis Control element. The requested state change was successful as evidenced by the zero Response Status.	→ Axis Control = 0	
The state change request transaction is now complete. Another Axis Control request can be processed at this time.	← Axis Response = 0 ← Response Status = 0	Drive sees a zero Request Code in the Axis Control element and zeroes the associated Acknowledge Code in the Axis Response element.

**Table 5-46.77 Unsuccessful Drive Control Request Cycle**

<b>Controller Action</b>	<b>Connection</b>	<b>Drive Action</b>
Controller sets Request Code in the Axis Control element to request a state change.	→ Axis Control = x	Drive sees non-zero Request Code, x, in Axis Control element and initiates the requested state change.
	← Axis Response = x ← Response Status = Error Code	If state change operation is unsuccessful, the Drive indicates the problem via a non-zero Response Status Error Code along with the Acknowledge Code matching the originating Request Code, x
Controller sees the non-zero Response Status associated with the Request Code, x, in the Axis Response, handles the error condition based on the Error Code, and clears the originating Request Code in the Axis Control element.	→ Axis Control = 0	
The state change request transaction is now complete. Another Axis Control request can be processed at this time.	← Axis Response = 0 ← Response Status = 0	Drive sees a zero Request Code in the Axis Control element and zeroes the associated Acknowledge Code in the Axis Response element.

Object state transitions can also be initiated by service requests such as the Test service requests. In general, these state change requests use services because they require one or more parameters to be passed to the device in order to initiate the state change.

## 5-46.32 State Behavior

The following section offers a detailed description of each of the states and state transitions of the Motion Axis Object state model.

### 5-46.32.1 Off State

This is the state of the Motion Axis Object with power off.

### 5-46.32.2 Self Test State

When power is applied to the device, or the device is reset, the device typically goes through a series of self-test diagnostics and internal device parameters are set to their power-up default values. Once completed successfully, the device and all its associated axis instances transition to the Initializing state and are ready for initialization by the associated controller. If unsuccessful, the device and all its associated axis instances transition immediately to the Major Faulted state by declaring an Initialization Fault that is classified as Un-recoverable. Clearing this fault can only be accomplished through a power cycle and is most likely the result of a device hardware problem..

### 5-46.32.3 Initializing State

During the Initializing state, network connections to the device are established by the controller. Once connections are established, the controller sends services to the device to set the Motion Axis Object configuration attributes to values stored in the controller. If the device supports synchronous operation, the controller then synchronizes with the device using the Group Sync service. Once this entire process has been completed successfully, the device and all its configured axis instances transition to the Pre-charge state. If a problem is found during this initialization process, an Initialization Fault is generated. An Initialization Fault is viewed as an unrecoverable fault so clearing the fault can only be accomplished through a power cycle or a device reset service to the associated Identity Object.

### 5-46.32.4 Pre-Charge State

The Pre-Charge state the device is waiting for the DC Bus to fully charge. Once the DC Bus reaches an operational voltage level, as indicated by the DC Bus Up bit in the Axis Status attribute, the device and all its configured axis instances transition to the Stopped state. Any attempt for the controller to enable an axis via the Axis Control mechanism while it is in the Pre-charge state is reported back to the controller as an error in the Response Status and the axis remains in the Pre-charge state.

### 5-46.32.5 Stopped State

In the Stopped state the device's inverter power structure should either be disabled and free of torque or held in a static condition via an active control loop. No motion can be initiated by the device in the Stopped State nor can the device respond to a planner generated command references. In general, the axis should be at rest, but if an external force or torque is applied to the load, a brake may be needed to maintain the rest condition. In the Stopped state, main power is applied to the device and the DC Bus is at an operational voltage level. If there are any Start Inhibit conditions detected while in this state, the axis transitions to the Start Inhibited state. If an Enable request or one of the Run Test service requests is applied to an axis in the Stopped state, the axis transitions to the Starting state.

### 5-46.32.6 Starting State

When an Enable request is given to an axis in the Stopped state, the axis immediately transitions to the Starting state. In this state, the device checks various conditions before transitioning the axis to the Running state. These conditions can include Brake Release delay time and Induction Motor flux level. The control and power structures are activated during the Starting state but the command reference is set to a local static value and does not track the command reference derived from the motion planner. If all the starting conditions are met, the axis state transitions to either the Running state or the Testing state.

### 5-46.32.7 Running State

The Running state is where the work gets done. In this state, the axis's power structure is active and the selected Control Mode is enabled and actively tracking command data from the controller based motion planner output to affect axis motion. The motion axis remains in the Running state until either a fault occurs or it is explicitly commanded to stop via a Axis Control request.

#### 5-46.32.8 Testing State

When any one of the Run Test request services is sent to the motion axis while in the Stopped state the axis immediately transitions to the Starting State, and then once the Starting conditions are met, the axis transitions to the Testing state. This Testing state is like the Running state in that the axis's power structure is active, but in the Testing state one of the axis's built-in test algorithms is controlling the motor, not a motion planner. In the Testing state the device excites the motor in various ways while performing measurements to determine characteristics of the motor and load. The motion axis remains in this state for the duration of the requested test procedure and then returns to the Stopped state. The motion axis can also exit the Testing state by either a fault or an explicit Axis Control request.

#### 5-46.32.9 Start Inhibited State

The Start Inhibited state is the same as the Stopped state with the exception that the axis has one or more "start inhibit" conditions that prevent it from successfully transitioning to the Starting state. These conditions can be found in the Start Inhibit Status attribute. Once corrected, the axis state automatically transitions back to the Stopped state.

#### 5-46.32.10 Stopping State

When a Disable request is issued to the Motion Axis Object in the Running or the Testing state, the axis immediately transitions to the Stopping state. In this state, the axis is in the process of stopping. There are a number of different Stopping Modes supported by the Motion Axis Object. Once the selected stop mode procedure has completed, the axis transitions to the Stopped state.

#### 5-46.32.11 Aborting State

When a Major Fault occurs while the axis is in either the Running or Testing states, the motion axis immediately transitions to the Aborting state. The Aborting state executes the appropriate stopping action as specified by the device vendor. Once the stopping procedure is complete the axis transitions to the Major Faulted state.

#### 5-46.32.12 Major Faulted State

The Major Faulted state is identical to the Stopped state (or, if a Shutdown fault action was initiated, the Shutdown state) with the exception that there are one or more Major Faults active. In other words, a Major Faulted axis is a Stopped (or Shutdown) axis with a Major Fault condition present. Since faults are latched conditions, a Fault Reset request from the controller is required to clear the fault and, assuming the original fault condition has been removed, the axis transitions to the Stopped (or Shutdown) state.

#### 5-46.32.13 Shutdown State

When a Shutdown request is issued to the device or a Shutdown fault action is executed by the device, the targeted axis transitions to the Shutdown state. In the case of a Shutdown request, the axis immediately transitions from whatever state it is currently in to the Shutdown state. The Shutdown state has the same basic characteristics of the Stopped state except that it can be configured to drop the DC Bus power to the device's power structure. Regardless of whether or not DC Bus power is disconnected, this state requires an explicit Shutdown Reset request from the controller to transition to the Pre-Charge state. If the axis is configured to keep the DC Bus power active while in the Shutdown state then the motion axis transitions through the Pre-Charge state to the Stopped state. The Shutdown state offers an extra level of safety against unexpected motion.

### **5-46.33 Fault and Alarm Behavior**

The Motion Axis Object's Fault and Alarm handling functionality addresses both the need for a large and ever-expanding number of specific faults and alarms, the need for programmable actions, and the need for timely reporting of those faults and alarms to the controller.

Additionally, no compromises are made to restrict the resolution of the reported faults and alarms, so that the controller always has access to the unique axis condition and a meaningful diagnosis. Numerous Fault and Alarm related attributes can be included in the fixed portion of the cyclic Device-to-Controller Connection so the controller can monitor the condition of the motion axis in real-time, without cumbersome polling.

The Axis Status attribute contains bits to indicate whether an alarm condition is present. The Axis State enumeration indicates when the axis has a major fault, which could be either a regular runtime Axis Fault or an Initialization Fault. Fault Codes are also provided to report the specific fault condition that led to the current Faulted state and covers both major and minor Faults. But before we go into detail on this, we must first carefully define the terms used to describe the Fault and Alarm functionality of the Motion Axis Object.

#### **5-46.33.1 Exceptions**

Exceptions are runtime conditions that the device continually checks that might indicate improper behavior of the motion axis or operation outside of an allowable range. An exception can result in an alarm, a minor fault, or a major fault, depending on how the associated Exception Action has been configured – an exception can even be configured to be ignored. Exceptions are automatically cleared by the device when the underlying exception condition is no longer present.

#### **5-46.33.2 Exception Actions**

For each exception, the motion axis can be programmed a variety of actions via the Exception Action attribute. Exception Actions range from generating a major fault that results in the stopping of the motion axis all the way to taking no action at all. The Axis Fault Status attribute allows the controller to have immediate access to any exceptions that have been configured to generate a major or minor fault. The Axis Alarm Status attribute allows the controller to have immediate access to any exceptions that have been configured to be reported as alarms.

#### **5-46.33.3 Alarms**

Alarms are runtime exception conditions for which the device is to take no action other than to report as an alarm. Alarms and warnings, therefore, are basically synonymous. On a given device product, some exception conditions may not be able to simply be reported as an alarm without any associated action; for example an IPM fault in which the power module automatically shuts off without software intervention. Alarm conditions are automatically cleared when the underlying exception condition is no longer present.

#### **5-46.33.4 Major Faults**

Major Faults are either initialization faults or runtime exception conditions that the device has been configured to regard as a major fault. If such a runtime fault occurs during an operational state, e.g. Running or Testing, it results in the device stopping (or aborting) all axis motion. Major Faults ultimately transition the axis state to the Major Faulted state. A Major Fault that results from an exception condition is latched, and does not clear when the exception condition clears. A fault can only be cleared with a Fault Reset service request from the controller. If the fault condition is classified as an “unrecoverable fault”, only a power cycle or a device reset can clear the fault condition.

### 5-46.33.5 Minor Faults

Minor Faults are exception conditions that the device has been configured to report to the controller as a fault but not take any direct action. This provides the controller an opportunity to perform an application specific fault action that may not be supported by the device as one of the defined exception actions. Like alarms, Minor Faults do not initiate a state change nor does a Minor Faulted state even exist; a Minor Fault allows the motion axis to continue operation in the state that it is presently in. But unlike alarms, a Minor Fault is latched, i.e. the fault does not clear when the exception condition clears. Both Major and Minor Faults can only be cleared with a Fault Reset service request from the controller. Since the motion axis cannot transition from the Stopped state to any operational state until the Minor Fault is cleared, a Minor Fault is considered a Start Inhibit and is indicated by the Minor Fault bit being set in the Start Inhibit Status attribute.

### 5-46.33.6 Initialization Faults

Initialization Faults are faults that are generated during the power-up or device reset procedure when the device detects a problem that prevents normal device operation. This could be a hardware or firmware problem detected as part of its self-diagnostic tests or a problem with the attribute configuration process. These faults are not sourced by exception conditions and, therefore, they do not have configurable actions. Examples of initialization faults are corrupted memory data, calibration errors, or firmware startup problems. Initialization Faults that result in the Faulted state cannot be cleared with a Fault Reset service, so any kind of motion is impossible in this state; only a power-cycle or a Device Reset has a chance of clearing this kind of fault.

### 5-46.33.7 Fault Codes

The Fault Code attribute is an enumeration that indicates the exception condition that generated the fault. This attribute does not change unless a Fault Reset service is initiated. The Initialization Fault Code attribute is an enumeration that indicates the initialization fault that led to the current Faulted state.

## 5-46.34 Start Inhibit Behavior

A Start Inhibit is a condition that inhibits the axis from starting, i.e. transitioning to the Starting state for enabled axis operation. This condition does not generate an exception if a start attempt is made. If the circumstances that led to the Start Inhibit are no longer present, the start inhibit condition is automatically cleared by the device, returning the axis to the Stopped State.

If the motion axis is in the Start Inhibit state it indicates that one or more conditions are present that prevent the axis from transitioning to enabled operation. The Start Inhibit Status attribute reports the specific condition that is inhibiting the axis.

The Start Inhibit Code attribute is an enumeration that indicates a condition that presently is causing the axis to be in the Start Inhibit state. If the original inhibiting condition is removed, this attribute automatically cleared or else set to another inhibiting condition if one is present.

## 5-46.35 Visualization Behavior

Motion Axis Object state behavior has a direct impact on motion device visualization components. These components range from bicolor LEDs to multi-character alphanumeric displays. This section defines how the Motion Axis Object states affect the behavior of these visualization components.

### **5-46.35.1 Module Status LED**

Motion Axis Object states have a relationship to the state behavior of the Identity Object of the CIP Motion device and to its associated Module Status LED. The table below maps the states of the primary Motion Axis Object instance to the appropriate states of the Identity Object. All CIP Motion compliant devices are required to support a Module Status LED.

For more information regarding the Identity Object state model, refer to the Identity Object.

### **5-46.35.2 Axis Status LED**

To further augment the visual information provided by the standard Module Status LED, the CIP Motion device profile also defines the behavior of a second LED. This, so called, Axis Status LED provides visual indication of, for example, whether or not the DC Bus is energized, whether the axis is enabled or disabled, and even provides indication of active alarms or minor fault conditions. This LED is also required by CIP Motion compliant devices unless the device is equipped with a multi-character alphanumeric display.

The Axis Status LED uses three colors that can be generated by a standard bicolor Red/Green LED, namely Red, Green, and Amber. Amber (or Yellow) is the color produced when both the Red and Green junctions of the bicolor LED are on. The general meaning of these three colors are as follows:

**Green** – indicates a normal power-up or operational state.

**Amber** – indicates the presence of an alarm or start inhibiting condition.

**Red** – indicates presence of some form of fault condition.

When both minor fault and alarm conditions are present, the minor fault indication takes precedence.

The normal power up or device reset, both the Module Status and Axis Status LEDs shall start in the Red state (under hardware control) while the device processor is booting and then switch to the Green state for approximately 1 second once the device begins executing its self test.

Ideally, both LEDs should be Red for approximately 1 second and then Green for approximately 1 second prior to transitioning to a Standby state indication. This Red-Green sequence confirms proper operation of the LED indicators.

**Table 5-46.78 Axis State Mapping to Identity Object with LED Behavior**

<b>Identity Object State</b>	<b>Module Status LED</b>	<b>Motion Axis Object State</b>	<b>Axis Status LED</b>
Nonexistent – Power Off	Off	Off	Off
Device Self-Testing	Flash Red/Green	Self Test	Flash Red/Green
Standby	Flashing Green	Initialization – Bus not Up	Off
		Initialization – Bus Up	Flashing Green
		Shutdown – Bus not Up	Off
		Shutdown – Bus Up	Flashing Amber <sup>1</sup>
		Pre-Charge - Bus not Up	Off
		Start Inhibit	Flashing Amber <sup>1</sup>
Operational	Solid Green	Stopped	Flashing Green <sup>1,2</sup>
		Stopping	Solid Green <sup>1,2</sup>
		Starting	Solid Green <sup>1,2</sup>
		Running	Solid Green <sup>1,2</sup>
		Testing	Solid Green <sup>1,2</sup>
Major Recoverable Fault	Flashing Red	Aborting	Flashing Red
		Major Faulted	Flashing Red
Major Unrecoverable Fault	Solid Red	Aborting	Solid Red
		Major Faulted	Solid Red

1. The Motion Axis Object and the Identity Object define minor fault conditions. While a minor fault does not affect the Module Status LED, it does affect the Axis Status LED. When a minor fault condition is detected a normally Solid Green LED indication changes to alternating Red-Green-Red-Green, a normally Flashing Green LED indication changes to alternating Red-Off-Green-Off, and a normally Flashing Amber indications changes to Red-Off-Amber-Off.

2. The Motion Axis Object also defines alarm conditions. When an alarm condition is detected, a normally Solid Green LED indication changes to alternating Amber-Green-Amber-Green while a normally Flashing Green LED indication changes to alternating Amber-Off-Green-Off.

### 5-46.35.3 Alphanumeric Display

In addition to the required LED visualization provided by the Module Status LED and Axis Status LED the Motion Axis Object also defines the behavior of an optional alphanumeric display to more explicitly indicate the condition of the device. Such a display is particularly useful for monitoring progress through the initialization process and providing detailed diagnostic information concerning fault, alarm, and inhibit conditions. Alphanumeric displays can range from simple seven-segment displays to multi-character alphanumeric displays.

### 5-46.35.4 Seven-Segment Display

If the device is equipped with a seven-segment display, the display can be used to indicate progress through the Initialization state and various fault, alarm, and inhibit conditions. At a minimum, the display should support the following mapping to various conditions of the device.

**Table 5-46.79 CIP Motion Device Seven-Segment Display Behavior**

Display Digit	Drive Condition
8	Executing drive Self-Test
0	Waiting for connection to controller.
1	Configuring device attributes.
2	Waiting for group synchronization.
3	Waiting for DC Bus to charge.
4	Device is operational
I##	Initialization Fault Code
IE##	Initialization Fault Code - Manufacturer extensions
F##	Axis Fault Code
FE##	Axis Fault Code - Manufacturer extensions
A##	Alarm Code
AE##	Alarm Code - Manufacturer extensions
S##	Start Inhibit Code
SE##	Start Inhibit Code - Manufacturer extensions

### 5-46.35.5 Multi-Character Alphanumeric Display

If the device is equipped with a multi-character alphanumeric display, even more useful device information can be conveyed to the user via scrolling or static character fields. The capabilities of the display generally dictate the length of the character strings that can be effectively displayed. Therefore, the CIP Motion Device Profile does not dictate exactly what is displayed. Nevertheless, for the purpose of enforcing consistent behavior among devices compliant with the CIP Motion Device Profile, the Motion Axis Object dictates exactly what is displayed for the conditions outlined in the table below.

**Table 5-46.80 CIP Motion Multi-Character Alphanumeric Display Behavior**

Display String	Device Condition
SELF-TEST	Executing device Self-Test
CONNECTING	Connecting (waiting for Forward_Open)
CONFIGURING	Configuring Motion Device Attributes
SYNCING	Synchronizing (waiting for GroupSync)
PRE-CHARGE	Waiting for DC Bus Up
SHUTDOWN	Axis has been Shutdown
STOPPED	Axis has stopped
INHIBITED	Axis is Start Inhibited
STARTING	Axis is Starting
RUNNING	Axis is Running
TESTING	Axis is executing a Test procedure
STOPPING	Decelerating to a stop as a result of a disable
ABORTING	Decelerating to a stop as a result of a fault
INIT FLT S##	Initialization Fault – Std and Fault Code
INIT FLT M##	Initialization Fault – Mfg and Fault Code
FLT S##	Fault – Std. and Fault Code
FLT M##	Fault – Mfg and Fault Code
ALARM S##	Initialization Fault – Std and Fault Code

**Motion Axis Object, Class Code: 42<sub>hex</sub>**

Display String	Device Condition
ALARM M##	Initialization Fault – Mfg and Fault Code
INHIBIT S##	Initialization Fault – Std and Fault Code
INHIBIT M##	Initialization Fault – Mfg and Fault Code

In addition to this basic functionality, the alphanumeric display can also provide detailed text descriptions of fault, alarm, and inhibit conditions. Definition of these specific strings is well beyond the scope of this object and is left to the discretion of the device vendor.

**5-46.35.6 Multi-Axis Device Visualization**

In the case where there are multiple motion axis instances supported by the device the above behavior needs further explanation.

First of all, there is only one Module Status LED per device node, so its condition is a roll-up of the states of all the Motion Axis Object instances. By contrast, one Axis Status LED is associated with each Motion Axis Object instance in the motion device that has a power structure. A single multi-character alphanumeric display can easily manage multiple motion axis instances.

In the case of a motion axis instance configured as a Master Feedback, there is very little state information to visualize; the motion axis is in the Stand-by state called Stopped and remains in that state unless there is a feedback fault that transitions it to Major Faulted. A Master Feedback axis, therefore, has no affect on the Module Status LED unless a fault occurs, in which case the Module Status shows Flashing Red (Major Recoverable Fault). A Master Feedback axis has no affect on the Axis Status LED nor is a separate LED required for such an axis.

In the case of a multiple motion axis instances, each with a separate power structure, the behavior of the Module Status LED is a roll-up of the states of the device axes. When the device axes are in disparate states, the Module Status LED condition is based on the following precedent.

1. Major Unrecoverable Fault
2. Major Recoverable Fault
3. Standby
4. Operational

In other words, as far as the Module Status LED is concerned a Major Recoverable Fault on Axis 1 has a higher precedence than Axis 2 that is in the Standby state, Start Inhibit. Note that minor faults and alarms are not recognized by the Module Status LED.

Multi-character displays easily handle multiple axis instances by adding a “X#” prefix to the display string to specify the associated motion axis instance number. For Master Feedback axis instances, no specific display string is shown unless the Master Axis has a fault or alarm condition.

**Table 5-46.81 Multi-Axis Multi-Character Alphanumeric Display Behavior**

Display String	Drive Condition
SELF-TEST	Executing device Self-Test
CONNECTING	Connecting (waiting for Fwd Open)
CONFIGURING	Configuring Motion Device Attributes
SYNCING	Synchronizing (waiting for Group Sync)
PRE-CHARGE	Waiting for DC Bus Up

**Motion Axis Object, Class Code: 42<sub>hex</sub>**

Display String	Drive Condition
X#:SHUTDOWN	Axis has been Shutdown
X#:STOPPED	Axis has stopped
X#:INHIBITED	Axis is Start Inhibited
X#:STARTING	Axis is Starting
X#:RUNNING	Axis is Running
X#:TESTING	Axis is executing a Test procedure
X# : STOPPING	Decelerating to a stop as a result of a disable
X# : ABORTING	Decelerating to a stop as a result of a fault
X# : INIT FLT S##	Initialization Fault – Std and Fault Code
X# : INIT FLT M##	Initialization Fault – Mfg and Fault Code
X# : FLT S##	Fault – Std. and Fault Code
X# : FLT M##	Fault – Mfg and Fault Code
X# : ALARM S##	Initialization Fault – Std and Fault Code
X# : ALARM M##	Initialization Fault – Mfg and Fault Code
X# : INHIBIT S##	Initialization Fault – Std and Fault Code
X# : INHIBIT M##	Initialization Fault – Mfg and Fault Code

X# = Axis Instance Number (0, 1, 2, 3 ...) → 0 = class instance

### 5-46.36 Command Generation Behavior

#### 5-46.36.1 Command Data Sources

Command data can take the form of Controller Position, Velocity, Acceleration, and Torque Commands. The command data element provided is specified by the Command Data Set attribute, which is based on the selected Control Mode. The primary command data element can be augmented by higher order command elements for the purposes of generating high quality feed-forward signals.

#### 5-46.36.2 Command Fine Interpolation

For synchronized, high-performance applications using CIP Motion, command data is received from the CIP Motion Controller-to-Device Connection and processed by a fine interpolator. The job of the fine interpolator is to compute coefficients to a trajectory polynomial that is designed to reach the command data value at its associated Command Target Time. Depending on the specific command data element, the trajectory can follow a 1<sup>st</sup>, 2<sup>nd</sup>, or 3<sup>rd</sup> order polynomial trajectory with initial conditions based on current axis dynamics. Since the polynomial is a function of time, a new fine command value can be calculated any time the CIP Motion Device needs to perform a control calculation. As a result, it is not necessary that the device's control calculation period be integrally divisible into the Controller Update Period.

To improve device interchangeability, the Motion Axis Object recommends a minimum order for the fine interpolators. Since contemporary motion planners typically generate their trajectories based on 3<sup>rd</sup> order polynomials in position, it is important that the fine interpolators reproduce these trajectories with high fidelity. Therefore, the position fine interpolator is defined as 3<sup>rd</sup> order, the velocity interpolator is 2<sup>nd</sup> order, and the acceleration and torque interpolators are both 1<sup>st</sup> order.

Position Fine Interpolation Polynomial:

$$P(t) = a_0 + a_1 * (t-t_0) + a_2 * (t-t_0)^2 + a_3 * (t-t_0)^3$$

Velocity Fine Interpolation Polynomial:

$$V(t) = b_0 + b_1 * (t-t_0) + b_2 * (t-t_0)^2$$

Acceleration Fine Interpolation Polynomial:

$$A(t) = c_0 + c_1 * (t-t_0)$$

Torque Fine Interpolation Polynomial:

$$T(t) = d_0 + d_1 * (t-t_0)$$

In these equations, time  $t_0$  represents the Command Target Time for the previous motion planner update such that when  $t = t_0$  the position, velocity, acceleration, and torque command values are equal to the values sent in the previous motion planner update, i.e.  $P_{-1}$ ,  $V_{-1}$ ,  $A_{-1}$ , and  $T_{-1}$ . This establishes the 0<sup>th</sup> order coefficients of the polynomials.

$$P(t_0) = P_{-1} = a_0$$

$$V(t_0) = V_{-1} = b_0$$

$$A(t_0) = A_{-1} = c_0$$

$$T(t_0) = T_{-1} = d_0$$

The higher order polynomial coefficients are calculated such that by the next motion planner update, corresponding to Command Target Time,  $t_1$ , the position, velocity, acceleration, and torque command values are the values sent in the latest motion planner update, i.e.  $P_0$ ,  $V_0$ ,  $A_0$ , and  $T_0$ .

$$P(t_1) = P_0$$

$$V(t_1) = V_0$$

$$A(t_1) = A_0$$

$$T(t_1) = T_0$$

Using the above polynomial interpolation equations, the CIP Motion device can compute position, velocity, acceleration, and torque command values at any time by plugging in the current System Time value of the CIP Motion device into the variable,  $t$ . This allows the device's control calculation to be performed according to a schedule that is independent of the controller's update schedule.

One thing that must be done, however, is to adjust the Command Target Time,  $t_0$ , should there be a shift in the System Time Offset for the device;  $t_0$  and  $t$  must always be based on the same System Time reference system. For example, assume the device's System Time Offset when the control command timestamp,  $t_0$ , was received is  $\text{Offset}_0$ . If the command interpolation equation is to be applied at  $t = t_1$  and the current System Time Offset is defined as  $\text{Offset}_1$  then  $t_0$  must be adjusted as follows before executing the polynomial:

$$\text{Adjusted } t_0 = t_0 * (\text{Offset}_1 - \text{Offset}_0)$$

The polynomial coefficients are computed based on standard formulas that are a function of the history of command values over the last few updates. The number of historical command values used in the formula depends on the order of the polynomial. For example, the third order command position polynomial uses the three previous command position values. For convenience, the interpolator polynomial coefficient formula's are as follows:

Position Fine Interpolation Polynomial Coefficients:

$$\begin{aligned}a_0 &= P_{-1} \\a_1 &= 1/T * (\Delta P_0 - 1/2 * \Delta V_0 + 1/6 * \Delta A_0) \\a_2 &= 1/T^2 * (1/2 * \Delta V_0 - 1/2 * \Delta A_0) \\a_3 &= 1/T^3 * (1/3 * \Delta A_0)\end{aligned}$$

Velocity Fine Interpolation Polynomial Coefficients:

$$\begin{aligned}b_0 &= V_{-1} \\b_1 &= 1/T * (\Delta V_0 - 1/2 * \Delta A_0) \\b_2 &= 1/T^2 * (1/2 * \Delta A_0)\end{aligned}$$

Acceleration Fine Interpolation Polynomial Coefficients (Torque is same form as Accel):

$$\begin{aligned}c_0 &= A_{-1} \\c_1 &= 1/T * \Delta A_0\end{aligned}$$

The above equations are based on the following nomenclature:

$$\begin{aligned}T &= \text{Controller Update Period} \\ \Delta P_0 &= (P_0 - P_{-1}) \\ \Delta V_0 &= (V_0 - V_{-1}) = (P_0 - 2P_{-1} + P_{-2}) \\ \Delta A_0 &= (A_0 - A_{-1}) = (V_0 - VP_{-1} + V_{-2}) = (P_0 - 3P_{-1} + 3P_{-2} + P_{-3})\end{aligned}$$

Note that when  $t > t_1$ , the fine interpolation polynomial becomes an extrapolation polynomial. In the absence of a fresh update from the motion planner, the extrapolation polynomial can be used to provide estimated command data to the control structure until fresh motion planner command data is available. Once fresh command data is made available new polynomial coefficients must be computed without delay. In this way, the motion control can “ride-thru” occasional late or lost connection data packets resulting in a robust distributed motion control network solution. To be clear, late connection data is always applied and never thrown away; late data still represents the freshest data available from the controller and the extrapolation polynomial insures that the command data is applied in such a way as to maintain a smooth motion trajectory despite variations in command data delivery.

When the update period of the motion planner is short enough relative to the dynamics of the command trajectory, or is comparable to the device’s control calculation period, fine interpolation may not be necessary. The motion planner can make this determination by comparing the planner update period to that of the device’s control calculation period. When fine interpolation is used, the planner must add additional planner update periods to the planner time stamp, so it is advantageous to eliminate this planner update period delay if interpolation is not necessary.

Even though fine interpolation may not be necessary in some cases, it does not mean that the command data is to be applied directly to the control structure. It still may be necessary to calculate the above polynomials so the device can extrapolate the command value when the device’s control update occurs. That is because, in general, the device’s control update time stamp does not need to match the time stamp of the command data.

Finally, there are CIP Motion device types and control modes that do not require the dynamic accuracy that time-stamped interpolation and extrapolation provide. Various velocity and torque control applications, for example, may fall in this category. In general, command data can also be applied to the control structures of Variable Frequency drives without interpolation or extrapolation.

### **5-46.36.3 Feed-Forward Signal Selection**

The fine interpolators defined as part of the Motion Axis Object can generate higher derivatives of the command data input to serve as feedforward signals. Superior signal quality, however, can be provided by the motion planner trajectory generators. The feedforward selection blocks pick the best feed-forward signal to apply based on the bits set in the Command Data Set attribute. The best signal is defined as the signal derived using the fewest differencing operations. Note that the interpolated command position is applied directly to the position control loop without any of the typical de-referencing and offsets. It is assumed that these operations are performed by the controller based motion planner.

### **5-46.36.4 Velocity Limiter**

The Interpolated Velocity Command signal passes through a series of dynamic limiters that are commonly used for velocity control. The Velocity Limiter provides independent positive and negative limits for the velocity command reference that can be applied to the device's control structure.

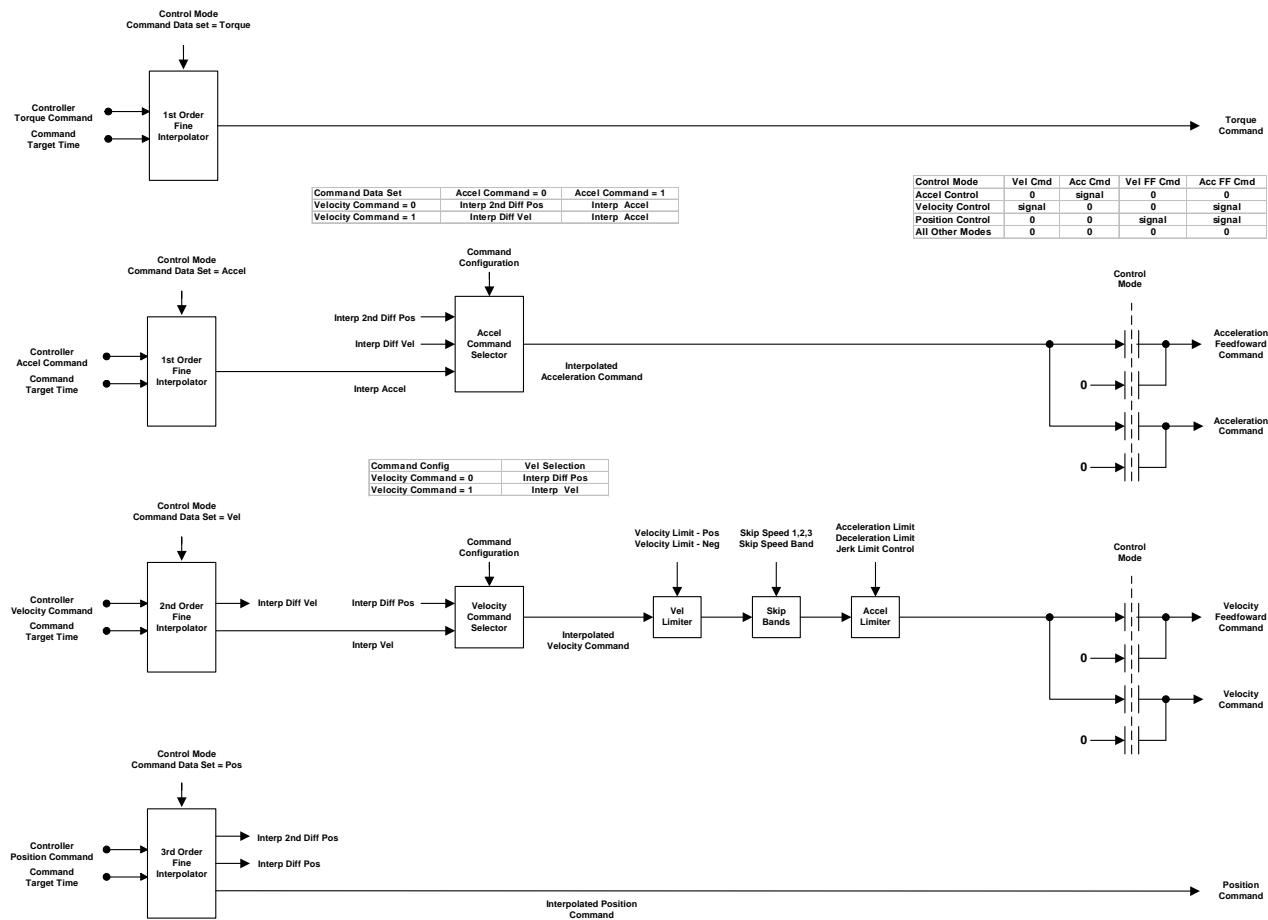
### **5-46.36.5 Skip Bands**

Skip Bands are typically used in Frequency Control applications when certain speeds excite mechanical resonance frequencies of the motor and load. The Skip Band feature allows three separate Skip Speeds to be defined that shift the Interpolated Velocity Command signal to avoid, or skip, these problematic speeds. The Skip Speed Band determines the range of speeds centered on the three Skip Speeds that the device avoids. If the Interpolated Velocity Command falls is within the Skip Band but below the Skip Speed the Velocity Command output is set to the Skip Speed, minus  $\frac{1}{2}$  the Skip Speed Band. If the Interpolated Velocity Command falls is within the Skip Band but above the Skip Speed the Velocity Command output is set to the Skip Speed, plus  $\frac{1}{2}$  the Skip Speed Band.

### **5-46.36.6 Acceleration Limiter**

The Acceleration Limiter limits the rate of change of the Velocity Command based on independent Acceleration and Deceleration Limits. Such a limiter is particularly valuable when the Interpolated Velocity Command is changed in discrete steps, such as when the motor speed is being controlled via a series of Controller Velocity Command set-points. The Acceleration Limiter turns the step changes of the Interpolated Velocity Command into controlled acceleration and deceleration ramps. The Acceleration Limiter can also support Jerk Limit Control that produces an S-Curve response to step changes in velocity command, providing even smoother speed transitions. The Jerk Limit Control attribute determines what percentage of the acceleration or deceleration ramps is S-Curve with the remaining portion of the ramp governed by the fixed Acceleration or Deceleration Limit.

Figure 5-46.32 Command Generator



## 5-46.37 Feedback Interface Behavior

### 5-46.37.1 Feedback Sources

Feedback signals defined by the Motion Axis Object can be derived from any of 4 different feedback interface channels. The two primary feedback channels employed by the various closed loop control modes are designated Feedback 1 and Feedback 2. This allows the control loops to operate with either a motor based feedback device that is typically attached to the Feedback 1 channel or a load-side feedback device that is connected to the Feedback 2 channel. Which feedback source is used by the loop is governed by the Feedback Configuration attribute.

Each feedback interface is capable of supporting a number of different feedback device types as enumerated by the Feedback Type attribute. The feedback interface output is the number of feedback counts that the feedback device has moved since the last time the device was sampled. If the feedback device is an absolute device, the feedback interface also determines the absolute position of the feedback device at power-up and communicates that value to the Feedback Accumulator to preset the accumulator.

#### **5-46.37.2 Feedback Accumulator**

The role of the Feedback Accumulator depends on the configured Feedback Mode which can be either Incremental or Absolute. If Incremental is selected the accumulator simply accumulates changes to the feedback count value, a 32-bit signed integer, with every device update. If Absolute is selected, the Feedback Accumulator works basically the same way as in Incremental mode. The only difference is the initialization of the accumulator at device power-up. In Incremental mode, the Feedback Accumulator is set to zero, while in Absolute mode, the accumulator is initialized to the absolute position of the feedback device. This allows for the recovery of absolute position through a power-cycle as long as power-off movement of the absolute feedback device is limited to  $\frac{1}{2}$  of the absolute feedback range of the device. There is no requirement to extend the absolute position range of the feedback device through non-volatile storage of the accumulator. This simple absolute feedback handling mechanism is due to the fact that the CIP Motion places the responsibility of extending the absolute position range of the axis, and establishing the absolute machine position reference, on the controller.

#### **5-46.37.3 Commutation Unwind and Offset**

Also connected to the Feedback 1 interface is an Electronic Unwind block. This block is designed to unwind, or modulo, the position accumulator output to generate a signal that is proportional to the electrical angle of a Permanent Magnet motor based on the Pole Count or Pole Pitch of the motor. To align this signal with the physical UVW windings of the motor rather than the zero of the feedback device, a configurable Commutation Offset is added prior to the Electrical Unwind block.

#### **5-46.37.4 Feedback Filtering**

A configurable low-pass IIR filter is defined by the Motion Axis Object for filtering the velocity and acceleration estimates for each feedback channel. These filters can be used to reduce the level of quantization noise associated with differencing digital feedback signals. The bandwidth of the velocity and acceleration IIR filters for each feedback channel are individually programmable.

Figure 5-46.33 Feedback Channels 1 and 2

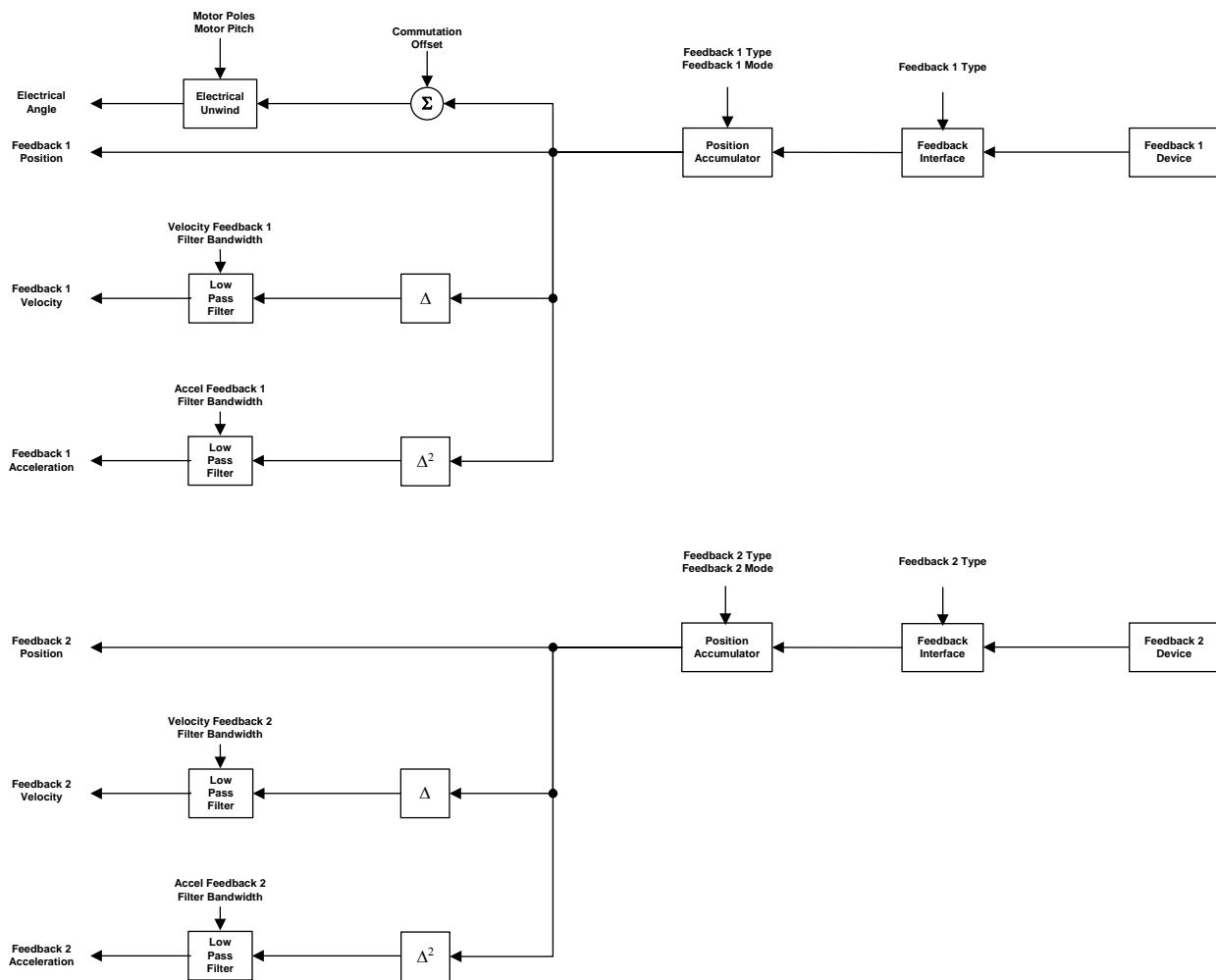
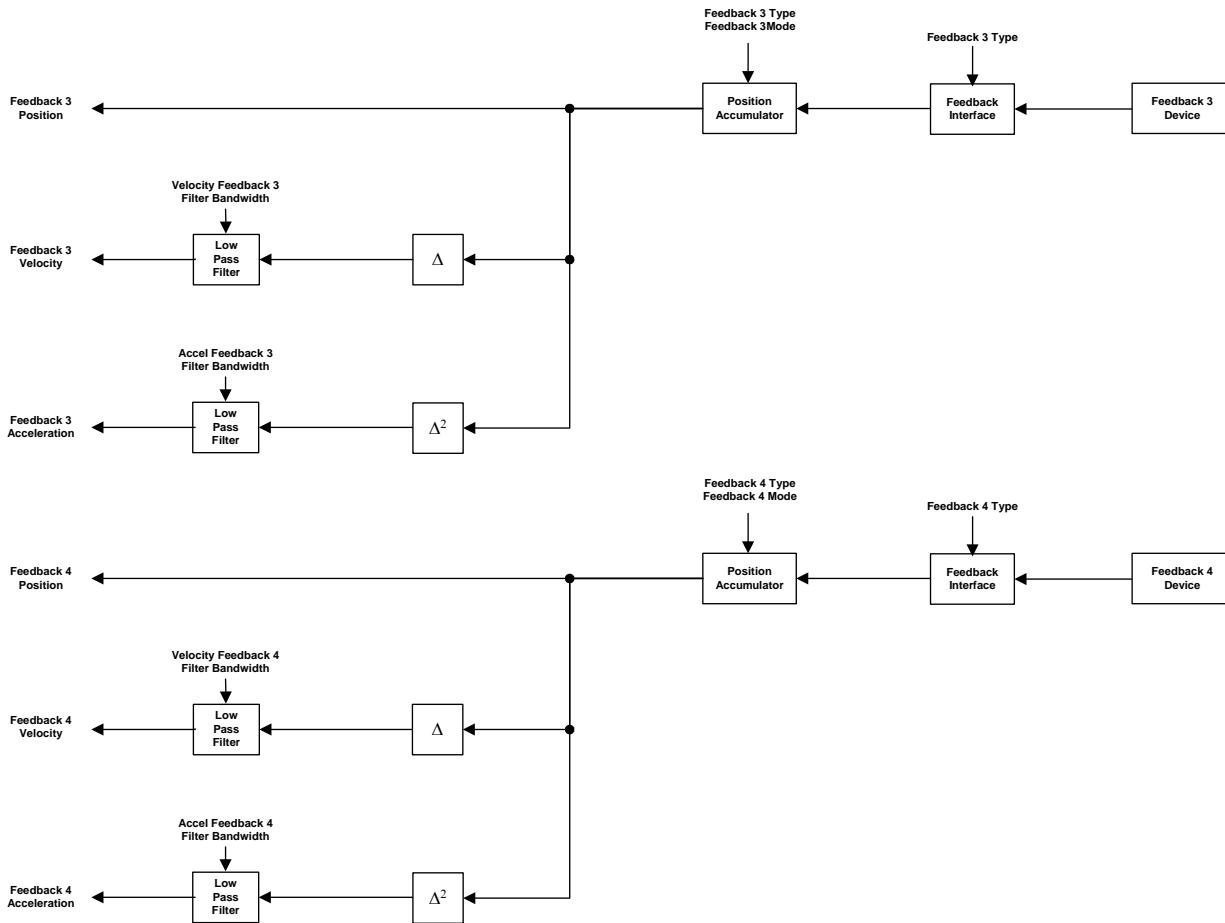


Figure 5-46.34 Feedback Channels 3 and 4



## 5-46.38 Event Capture Behavior

### 5-46.38.1 Event Input Sources

The Motion Axis Object defines a mechanism to capture both the feedback position and time stamp associated with specific state transitions of selected event input sources. Event input sources currently supported by the object are Registration 1, Registration 2, Marker, and Home Switch. These 4 event input sources apply to each supported feedback channel.

### 5-46.38.2 Event Latches

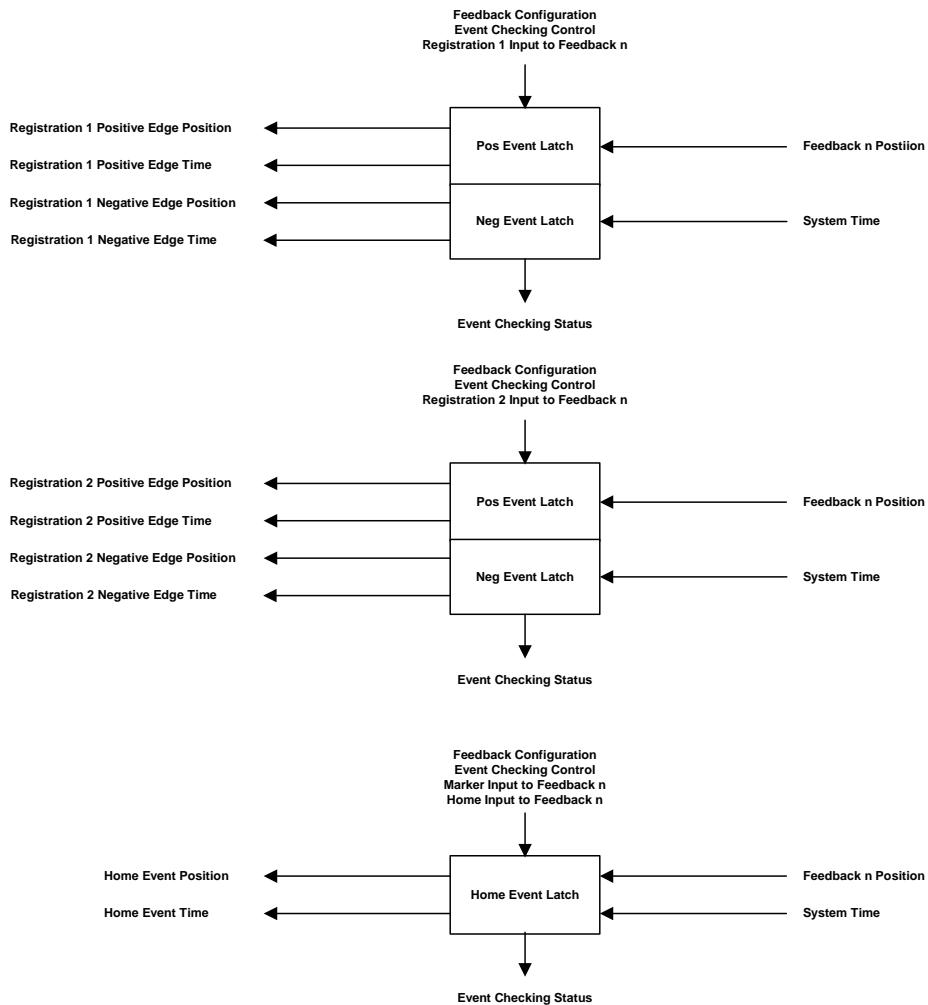
To facilitate accurate capture of both feedback position and time, hardware event latches are typically implemented as shown in the following block diagram. Note that two independent latches are defined for each registration input, one latch to capture positive edge transition events and one to capture negative edge transition events. This design enables capture of both registration events in applications with narrow registration pulses where the rising and falling edges occur nearly simultaneously. In addition to the registration latches, a separate latch is also defined for the home event capture. The home input event that triggers the Home Event Latch can be any of a number of different combinations of home switch and marker input events, i.e. marker transitions, switch transitions, or switch transitions followed by a marker transition.

With hardware based event latches, event capture accuracy is, in general, only limited by the latency of the associated event input. Registration and Marker event inputs are lightly filtered so event capture accuracy is on the order of 1  $\mu$ sec. In terms of position capture accuracy, that would be calculated as the product of the event capture accuracy and the speed of the axis. Home switch inputs are heavily filtered, in general, and therefore limited to an event capture accuracy of 1 to 10 msec. Thus, to get an accurate position capture based on a home switch input transition, a homing sequence with a slow homing speed is required.

### 5-46.38.3 Event Time Stamps

Since the registration time stamp is passed to the controller as part of the Event Notification data, the controller can apply the event time stamp to the position history of other axes in the system to interpolate their positions. This is particularly useful in applications where it is necessary to determine the location of several axes at the time of a single registration event. The more accurate the time stamp, the more accurately the controller can determine these positions.

**Figure 5-46.35 Event Capture Functionality**



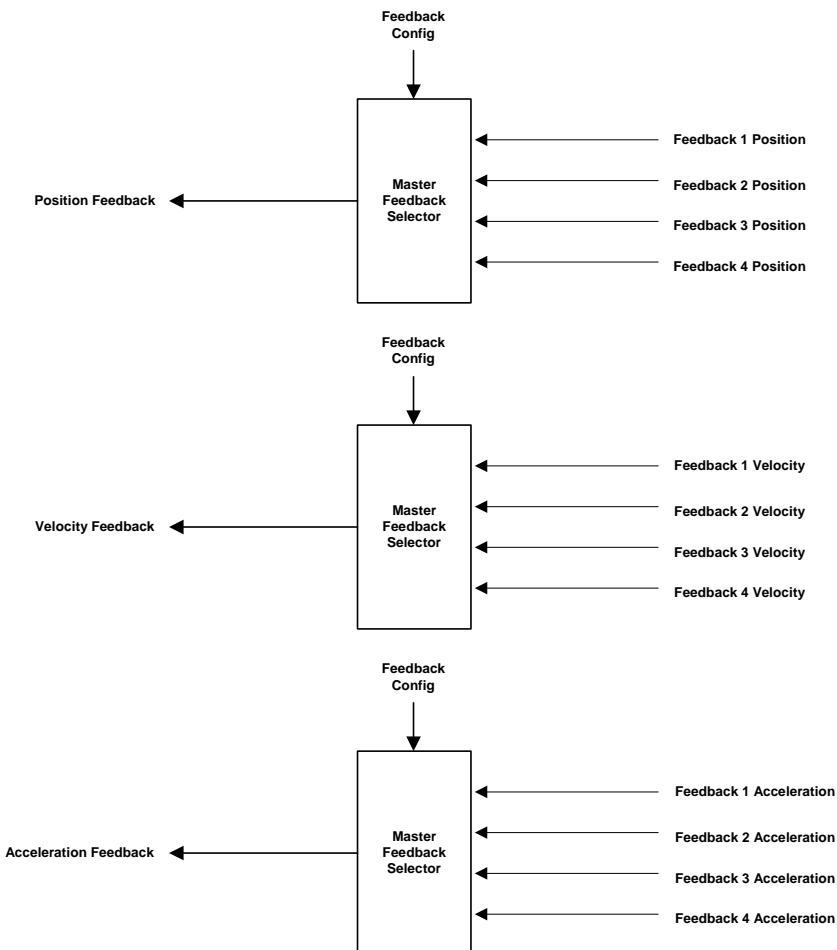
### 5-46.39 Control Mode Behavior

The instance attributes defined in Section 5-46.7 affect device behavior in the context of the Control Mode, Control Method, and Feedback Configuration. As discussed in 5-46.2.1 concerning the scope of the Motion Axis Object, there are basically 4 Control Modes common to CIP Motion devices, position control, velocity control, torque control, and no control. To this list we add a new control mode, acceleration control, to complete the progression from velocity control to torque control. This section provides a block diagram for each of the control modes in an effort to further define the collective behavior of the various Motion Axis Object attributes.

### 5-46.40 No Control (Feedback Only) Mode

A Motion Axis Object instance can be configured for No Control mode to allow for the position, velocity, and acceleration of any one of four possible feedback channels to be accessed by the controller via the Device-to-Controller Connection. These signals can then be distributed across the motion control system as a master axis for gearing and camming operations. In this mode, the Feedback Configuration attribute determines which feedback channel produces the Position, Velocity, and Acceleration Feedback signals.

**Figure 5-46.36 No Control (Feedback Only)**



#### **5-46.41 Position Control Mode**

In Position Control mode, the only operative Control Method supported by the object currently is Closed Loop servo control. At a later date the object could be expanded to include a Stepper based position Control Method.

#### **5-46.42 Closed Loop Position Control**

When performing closed loop position control, the CIP Motion device applies the Position Command signal output of the Command Generator to the position loop-summing junction. In addition to the Position Command, a Position Trim input is provided which can be used to provide an offset to the position loop. The classic PI control loop generates a Position Loop Output signal to an inner velocity loop.

##### **5-46.42.1 Position Feedback Configuration**

Feedback to the PI regulator can be derived from two different feedback channels. This flexibility allows the position loop to operate with either a motor-based feedback device that is typically attached to the Feedback 1 channel or a load-side feedback device that is connected to the Feedback 2 channel. Which feedback source is used by the loop is governed by the Feedback Configuration attribute.

When the Feedback Configuration calls for Dual Feedback operation, the position loop utilizes the Feedback 2 channel and the velocity loop uses the Feedback 1 channel. Since the two feedback channels may not have the same feedback resolution, it is necessary to convert position loop output from Feedback 1 counts to Feedback 2 counts prior to applying the output to the velocity loop summing junction. This is done scaling the position loop output by the Feedback 1/2 Count Ratio.

##### **5-46.42.2 Position PI Gains**

The Proportional Gain of the classic PI controller sets the unity gain bandwidth of the position loop in radians/sec, while the Integral Gain is used to drive the Position Error signal to zero to compensate for the effect of any static and quasi-static torque or forces applied to the load.

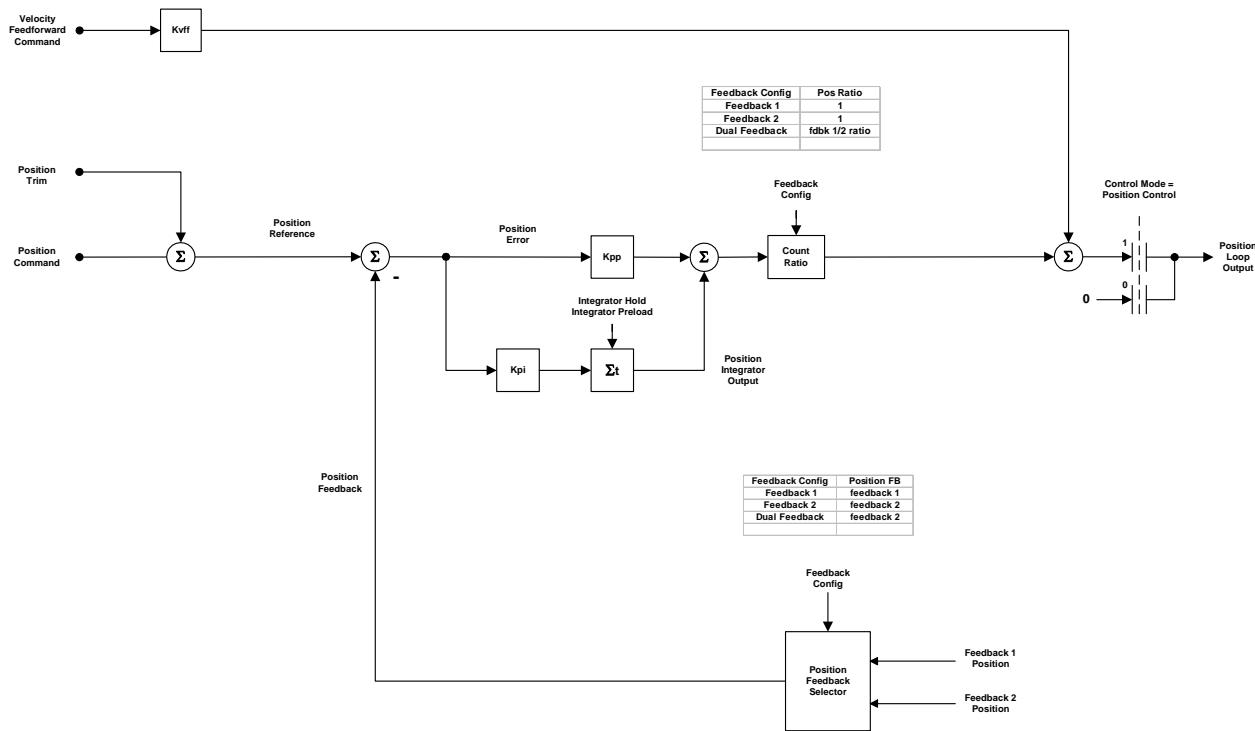
##### **5-46.42.3 Velocity Feed-Forward**

The inner velocity loop requires a non-zero command input to generate steady-state axis motor velocity. To provide the non-zero output from the CIP Motion device to the motor, a non-zero position loop output is required, which translates to a non-zero position error. This dynamic error between command position and actual position while moving is often called “following error”. Most closed loop motion control applications desire zero following error -- all the time! This could be achieved to some extent through use of the position integral gain control as described above, but typically the response time of the integrator action is too slow to be effective in high-performance motion control applications. An alternative approach that has superior dynamic response is to use Velocity Feedforward.

The Velocity Feedforward feature is used in Position Control mode to provide the bulk of the Velocity Reference input necessary to generate the desired motor velocity. It does this by scaling the Velocity Feed-forward Command signal output of the Command Generator by the Velocity Feedforward Gain and adding to the Position Loop Output generated by the position loop to form the Velocity Reference signal. With this feature, the position loop does not need to generate much effort to produce the required velocity command level; hence the Position Error value is significantly reduced. The Velocity Feedforward Gain allows the following error of the position control loop to be reduced to nearly zero when running at a constant velocity. This is important in applications such as electronic gearing and synchronization applications where it is necessary that the actual axis position not significantly lag behind the commanded position at any time.

The optimal value for Velocity Feedforward Gain is 100% theoretically. In reality, however, the value may need to be tweaked to accommodate velocity loops with finite loop gain. One thing that may force a smaller Velocity Feedforward value is that increasing amounts of feedforward tends to exacerbate axis overshoot. For this reason feed-forward is not recommended for point-to-point positioning applications.

**Figure 5-46.37 Closed Loop Position Control**



#### 5-46.43 Velocity Control Mode

In Velocity Control mode, there are two operative control methods supported by the object, Closed Loop Velocity Control and Open Loop Frequency Control.

**5-46.44 Closed Loop Velocity Control**

The Closed Loop velocity control method is targeted for applications that require tight speed regulation. Note that the command input to the velocity loop can be derived directly from the Velocity Command of the Command Generator when configured for Velocity Control Mode or from the Position Loop Output when configured for Position Control Mode as described in the previous section.

When serving as an outer velocity loop in Velocity Control Mode, the device applies the Velocity Command input to the velocity-summing junction as the Velocity Reference of a classic PI regulator. Also contributing to the Velocity Reference input signal is the Velocity Trim input, which can be used in conjunction with an outer control loop to make minor adjustments to the velocity of the motor.

When serving as an inner velocity loop in Position Control Mode, the device applies the Position Loop Output signal to the input of the velocity-summing junction. Input signals that are not applicable to the configured control mode are generally set to zero.

**5-46.44.1 Velocity Feedback Configuration**

Feedback to the PI regulator can be derived from either of the two available feedback transducers, Feedback 1 or Feedback 2. Which feedback source is used by the loop is governed by the Feedback Configuration enumeration.

**5-46.44.2 Velocity Error Filter**

A low pass filter can be optionally applied to the velocity error signal generated by velocity loop summing junction. The output of this filter becomes the Velocity Error signal that is subsequently operated on by the velocity loop PI control algorithm. When used, the filter is typically set between 5 to 10 times the velocity loop bandwidth. It is recommended that this filter be a two pole IIR filter to maximum its effectiveness at quantization noise filtering.

**5-46.44.3 Velocity PI Gains**

The velocity loop generates a Velocity Loop Output signal to the next inner loop via a classic PI control loop structure. The Proportional Gain of the controller sets the unity gain bandwidth of the velocity loop in radians/sec, while the Integral Gain is used to drive the Velocity Error signal to zero to compensate for any static and quasi-static torque or forces applied to the load.

The integral section of the velocity regulator includes an anti-windup feature. The anti-windup feature automatically holds the regulator's integral term when a limit condition is reached in the forward path. The anti-windup feature is conditioned by the arithmetic sign of the integrator's input. The integrator is held when the input's sign is such that the integrator output moves further into the active limit. In other words, the integrator is allowed to operate (not held) when the input would tend to bring the integrator output value off the active limit.

The integrator may also be configured for integrator hold operation. When the Integrator Hold attribute is set true, the regulator holds the integrator from accumulating while the axis is being commanded to move. This behavior is helpful in point-to-point positioning applications.

An automatic preset feature of the velocity regulator's integral term occurs when a transition is made from a torque control mode to speed control mode, using the Control Mode selection parameter. Upon transition to speed mode, the speed regulator's integral term is preset to the motor torque reference parameter. If the speed error is small, this provides a 'bumpless' transition from the last torque reference value present just prior to entering speed mode.

#### **5-46.44.4 Velocity Droop**

Another feature of the velocity regulator is the velocity droop function. The velocity error input to the integral term is reduced by a fraction of the velocity regulator's output, as controlled by the droop gain setting, Kdr. As torque loading on the motor is increased, actual motor speed is reduced in proportion to the droop gain. This is helpful when some level of compliance is required due to rigid mechanical coupling between two motors.

#### **5-46.44.5 Acceleration Feed-Forward**

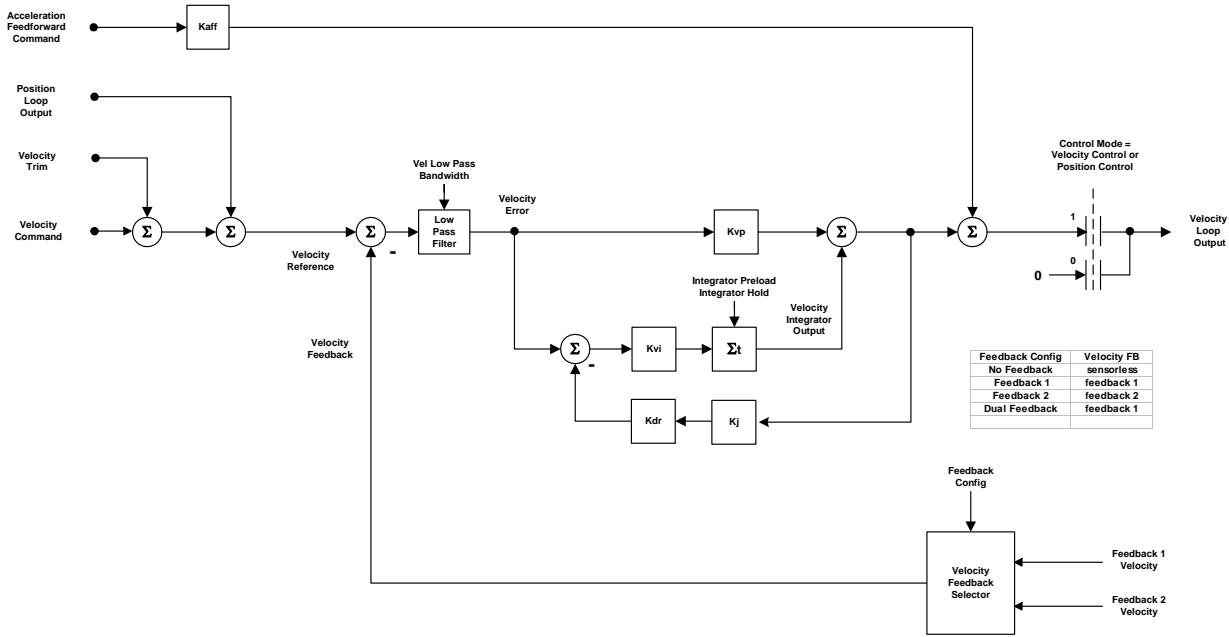
The velocity loop requires a non-zero velocity loop output to generate steady-state axis motor acceleration. To provide the non-zero output from the drive device to the motor, a non-zero velocity error is generally required. In position control applications this non-zero velocity error translates to a non-zero position loop error. Since many closed loop motion control applications require near zero control loop error, this behavior is not desirable. Again, the position and velocity loop error could be reduced by applying the velocity integral gain control as described above, but the integrator action is still too slow to be very effective. The preferred approach with superior dynamic response is to use Acceleration Feedforward.

The Acceleration Feed-forward Gain attribute is used to generate the output necessary to generate the commanded acceleration. It does this by scaling the Acceleration Feed-forward Command generated by the Command Generator by the Acceleration Feedforward Gain and adding it as an offset to the output of the velocity loop. With this feature, the velocity loop does not need to generate much control effort, thus reducing the amount of control loop error.

The optimal value for Acceleration Feedforward is 100% theoretically. In reality, however, the value may need to be tweaked to accommodate variations in load inertia and the torque constant of the motor. Like Velocity Feedforward, Acceleration Feedforward can result in overshoot behavior and therefore should not be used in point-to-point positioning applications.

When used in conjunction with the Velocity Feedforward, the Acceleration Feedforward allows the following error of the position or velocity control loop to be reduced to nearly zero during the acceleration and deceleration phases of motion. This is important in tracking applications that use electronic gearing and camming operations to precisely synchronize a slave axis to the movements of a master axis.

Figure 5-46.38 Closed Loop Velocity Control



## 5-46.45 Open Loop Frequency Control

Another Velocity Control method is the open loop Frequency Control method associated with so called Volts/Hertz or Variable Frequency Drives (VFDs) that do not have a current control loop and typically drive an induction motor. Velocity control with this method is achieved by controlling the voltage and frequency output of the drive device in some manor where voltage is generally proportional to frequency. For an induction motor, the velocity of the motor is determined by the Output Frequency of the drive device divided by the Motor Pole count. This control method is applicable to velocity control applications that do not require tight speed regulation and therefore do not require a feedback device. The following block diagram illustrates this open loop velocity control method.

### 5-46.45.1 Basic Volts/Hertz Operation

The Motion Axis Object provides a number attributes that are used to specify the relationship the CIP Motion device uses between output frequency (speed) and output voltage for a given (induction) motor. The Break Frequency and Break Voltage attributes define the point on the Volts/Hertz curve below which the Start Boost feature is applied. As the name indicates, Start Boost is used to provide a non-zero output voltage to the motor at standstill to assist start-up. The contribution of the Start Boost to the output voltage of the drive device tapers off to zero when the motor reaches the Break Frequency. Above the break point, output voltage and output frequency follow a linear slope to the point defined by the Motor Rated Frequency and Motor Rated Voltage. From this point on, the Volts/Hertz curve follows another linear slope to the point defined by the Max Frequency and Max Voltage attributes. This segment of the Volts/Hertz curve allows for operation above the rated frequency and voltage of the motor in applications where that is required.

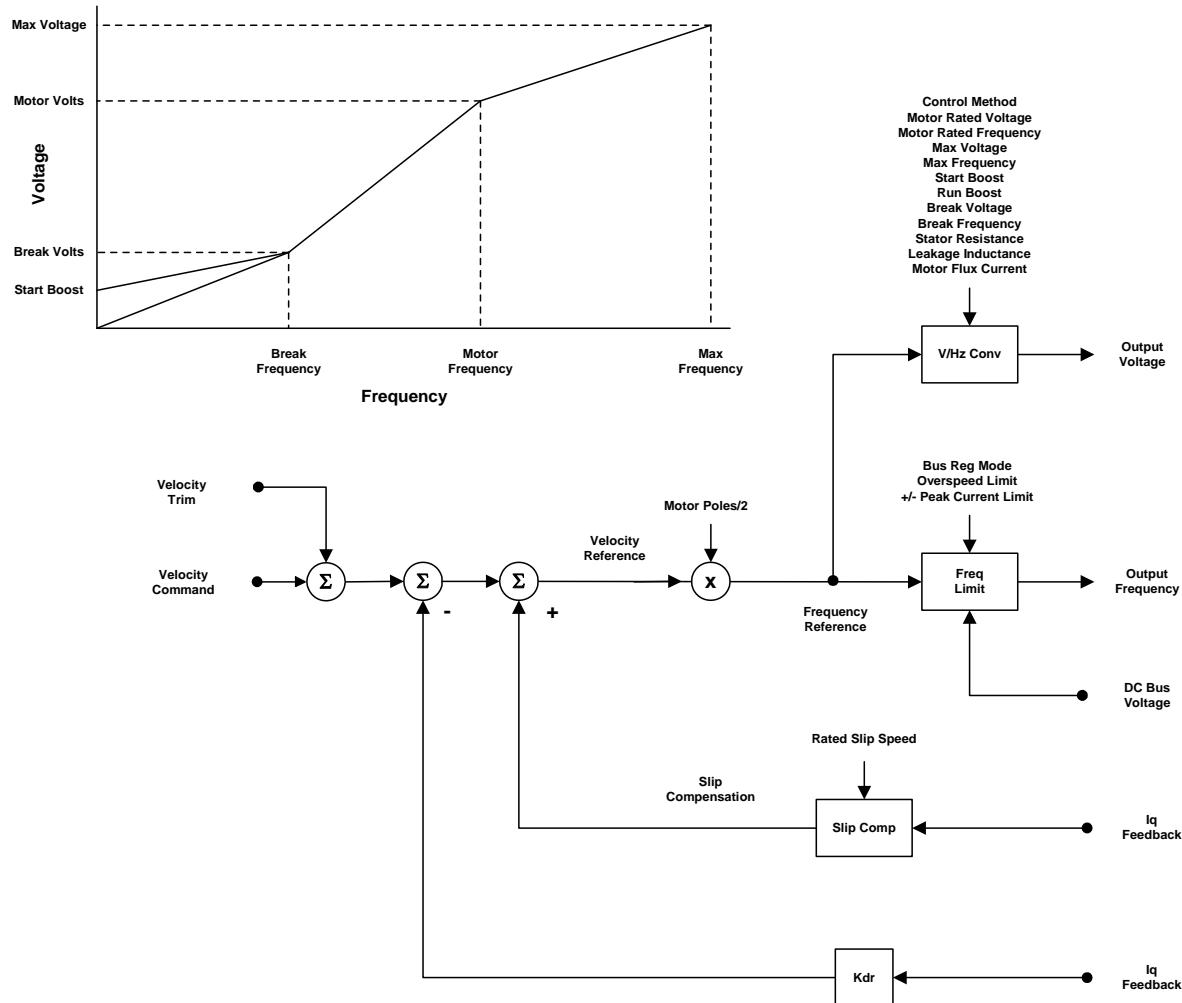
### 5-46.45.2 Slip Compensation

When driving an induction motor at a specific frequency, the actual motor velocity is generally less than the command speed, given by the output frequency divided by the motor pole count, by an amount that is proportional to the load torque applied to the motor. This difference in speed is called “Slip” and is a configuration attribute associated with the motor. The Motion Axis Object supports a Slip Compensation feature that is common to Variable Frequency Drives. The amount of Slip Compensation applied to the Velocity Reference is the product of the measured torque producing current,  $I_q$ , and the configured Rated Slip Speed

### 5-46.45.3 Velocity Droop

Another feature defined for the Frequency Control method is the droop function. The droop function reduces the velocity reference by a scaled fraction of the torque producing current,  $I_q$ , as controlled by the droop gain setting,  $K_{dr}$ . As torque loading on the motor is increased, actual motor speed is reduced in proportion to the droop gain. This is helpful when some level of compliance is required when performing torque sharing between two motors on a common load.

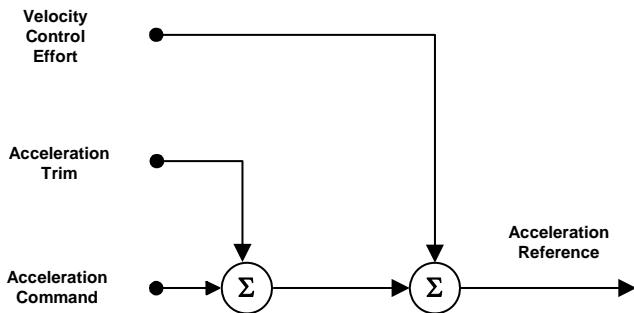
**Figure 5-46.39 Open Loop Frequency Control**



#### 5-46.46 Acceleration Control Mode

While dynamic motor control via an acceleration command is not common in the industry, Acceleration Control was added to the Motion Axis Object to complete the dynamic progression from velocity control to torque control. The output of the velocity loop, Velocity Loop Output, also has units of acceleration. So, like the other control modes we sum the contributions of the Acceleration Command, Acceleration Trim, and Velocity Loop Output to form the Acceleration Reference signal that serves as one of the primary inputs to the Torque Control section.

**Figure 5-46.40 Acceleration Control**



#### 5-46.47 Torque Control Mode

Torque is generally proportional to acceleration and to the torque producing motor current,  $I_q$ . The purpose of the Torque Control structure is to combine input signals to create a Torque Reference from a number of different sources based on the Control Mode and apply various filters and compensation algorithms to the Torque Reference to create a Filtered Torque Reference. The Filtered Torque Reference signal is scaled by the torque constant of the motor to become the Current Command input to the current loop. Since the motor current is also per unitized to the % Rated current of the motor, the torque constant,  $K_t$ , is nominally 1.

##### 5-46.47.1 Torque Input Sources

The Torque Control section can take input from a variety of sources depending on the Control Mode. Input to the Torque Reference path can come via the cyclic Torque Command or Torque Trim signal in Torque Control mode. In Position or Velocity Control mode, torque input can also be derived from the outer velocity loop by bringing in the Velocity Loop Output into the torque loop-summing junction. In Acceleration Control Mode, the Acceleration Reference signal is applied representing the sum of the cyclic Acceleration Command and Acceleration Trim data.

##### 5-46.47.2 Torque Scaling

Since the Velocity Loop Output and Acceleration related values have units of acceleration, these values are multiplied by a Torque Scaling gain,  $K_j$ , to convert to “per unitized” torque. The torque units defined by the Motion Axis Object are expressed as % of Rated Torque of the motor. The Torque Scaling gain, when properly configured, represents the total inertia or mass of the system that includes the motor and the load.

### 5-46.47.3 Low Pass Filter

The Low Pass Filter is effective in resonance control when the natural resonance frequency is much higher (>5x) than the control loop bandwidth. This filter works by reducing the amount of high-frequency energy in the output signal that excites the natural resonance. The Low Pass Filter design can be single pole or multiple poles. Care must be taken, however, to limit the amount of phase lag introduced by this filter to the control loop to avoid potential instability.

### 5-46.47.4 Notch Filter

The notch filters is effective in resonance control when the natural resonance frequency is higher than the control loop bandwidth. Like the Low Pass filter, the notch filter works by significantly reducing the amount of energy in the output signal that can excite the natural resonance. It can be used even when the natural resonance frequency is relatively close to the control loop bandwidth. That is because the phase lag introduced by the notch filter is localized around the notch frequency. For the notch filter to be effective, the Notch Frequency has to be set very close to the natural resonance frequency of the load.

A typical equation for the notch filter is as follows:

$$G(s) = \frac{s^2 + wn^2}{s^2 + s * wn/Q + wn^2}$$

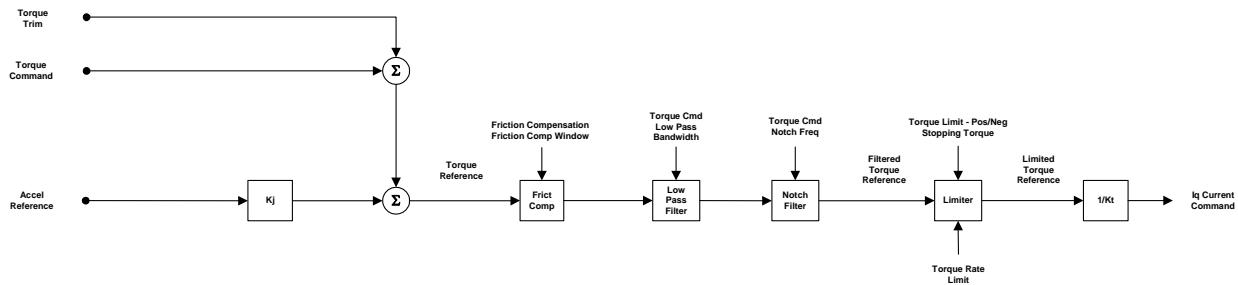
In this equation, Q represents the sharpness of the notch. In most implementations, the sharpness, Q, is typically hard-coded. The attenuation depth of the notch filter is infinite.

### 5-46.47.5 Torque Limiter

The Filtered Torque Reference signal passes through a limiter block to produce the Limited Torque Reference signal. The Torque Limiter block applies a torque limit to the signal that is based on the sign of the torque reference signal input and the state of the axis. During normal operation it is the Torque Limit – Positive and Torque Limit – Negative attributes, set by the user, is applied to the torque reference signal. When the axis is commanded to stop as part of a disable request or major fault condition the device applies the Stopping Torque Limit.

Also included with the torque limit block is a built in Torque Rate of Change Limit. This feature limits the rate of change of the torque reference output.

**Figure 5-46.41 Torque Control**



## 5-46.48 Current Control Mode

In general, motor torque is controlled by controlling the orientation and magnitude of the motor stator current vector with respect to the rotor magnetic flux vector. The Current Control loop is responsible for providing this control and is actually composed of two PI loops, one that controls the torque producing current,  $I_q$ , and one that controls the flux producing current,  $I_d$ . It is the quadrature component of current,  $I_q$ , that is used for dynamic torque control.

In the case of an induction motor, the flux producing current,  $I_d$ , is solely responsible for generating rotor flux. In the case of permanent magnet motors, rotor flux is generated by the rotor magnets and  $I_d$  is only used to in some cases to extend the speed range of the motor by changing the angle of stator field relative to the rotor field. In this case, the angle of  $I_q$  relative to the rotor field remains the same, i.e. at quadrature. But since the vector combination of  $I_q$  and  $I_d$  determines the stator flux angle relative to the rotor, increasing amounts of  $I_d$  can shift the stator flux away from quadrature to extend the speed range of the motor at the expense of torque.

### 5-46.48.1 Current Vector Limiter

The  $I_q$  Current Command passes through a Current Vector Limiter block before becoming the  $I_q$  Current Reference signal. This limiter block computes the combined vector magnitude of the  $I_q$  Current Command and the  $I_d$  Current Reference signals. The resultant current vector magnitude is compared to the Operative Current Limit that represents the minimum current limit from among a set of potential current limits of the drive device and motor. If the vector magnitude exceeds the Operative Current Limit, the  $I_q$  Current Command is reduced so the vector magnitude equals the Operative Current Limit. Potential current limit sources can be the Peak Current Limit ratings as well as the Thermal Limits for the Motor and Drive Inverter. Some of these limits are conditional and dynamic, such as the Motor and Inverter Thermal Current Limits derived from the thermal models for these devices. These limits are only active when the corresponding Motor and Inverter Overload Action attributes are set to provide current fold-back. The thermal current limits in this case would decrease as the simulated temperature of the modeled devices increases. The Bus Regulator Limit is only applied when the motor is regenerating power onto the DC Bus and is based on the Bus Regulator Power Limit.

With all these potential current limit sources that could be operative, a Current Limit Source attributes was included with the Motion Axis Object to identify the source of the active current limit.

### 5-46.48.2 Voltage Output

The output of each current loop is scaled by the motor inductance to generate a voltage command to the vector transformation block. It is the job of the vector transformation block to transform the torque producing,  $V_q$ , and flux producing,  $V_d$ , command signals from the rotating synchronous reference frame to the stationary stator reference frame. The resultant U, V, and W Output Voltage values are then applied to the motor by Pulse Width Modulation (PWM). The PWM Frequency is also a configurable attribute of the Motion Axis Object.

The magnitude of the  $V_q$ ,  $V_d$  vector is calculated in real time as the total Output Voltage signal. The maximum Output Voltage signal that can be applied to the motor is ultimately limited by the DC Bus Voltage and enforced by the Voltage Vector Limiter. Any attempt to exceed this value results in an Inverter Voltage Limit condition.

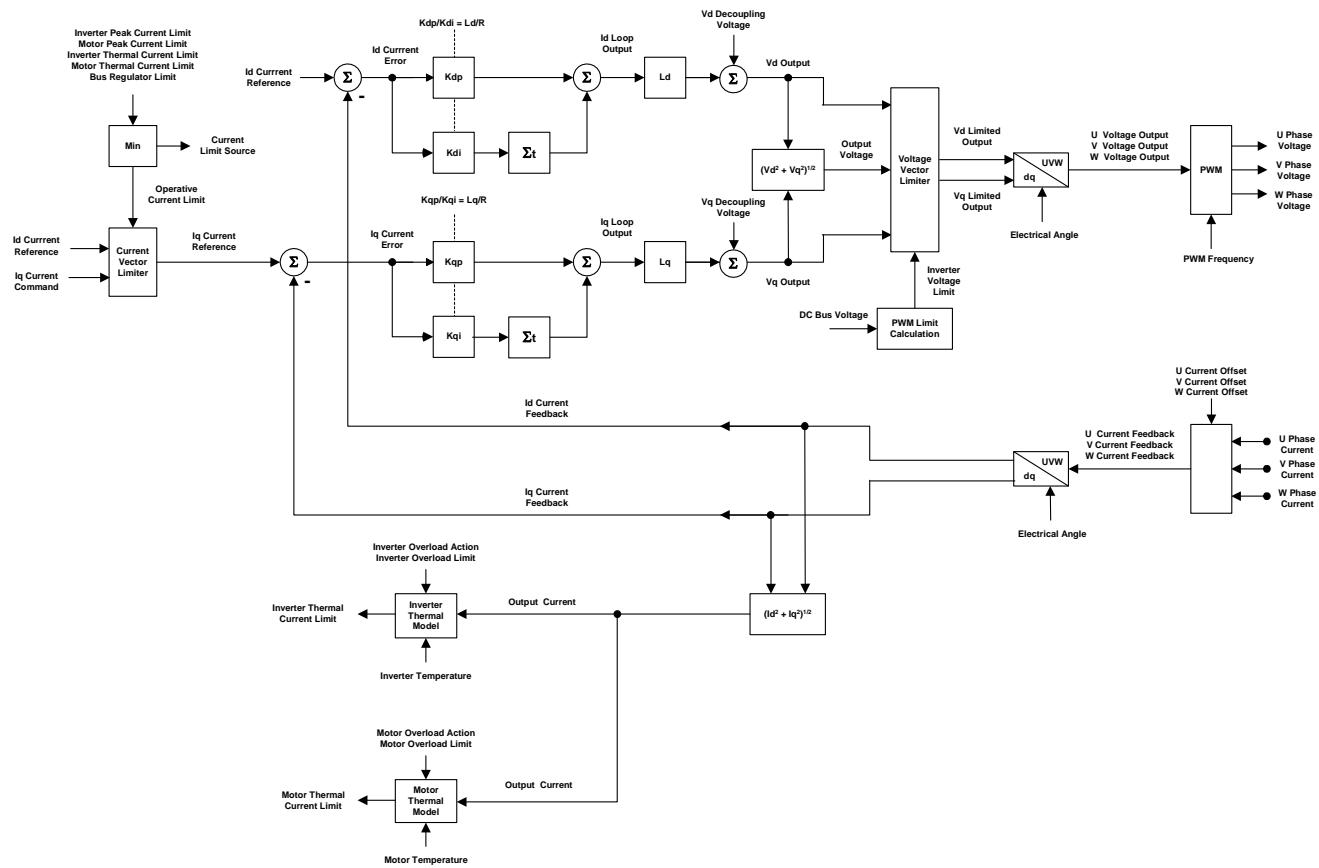
### 5-46.48.3 Current Feedback

Current feedback signals to the current loop are provided by two or three current sensors. The signals from these sensors are conditioned and corrected for device specific offsets to become the U, V, and W Current Feedback signals associated with the stationary motor stator frame. These three signals are transformed back to the synchronous reference frame to generate the I<sub>q</sub> and I<sub>d</sub> Current Feedback signals. The magnitude of the I<sub>q</sub>, I<sub>d</sub> current vector is calculated in real-time and used as an input to the thermal models for the inverter and motor.

### 5-46.48.4 Motor Commutation

Motor commutation is critical to closed loop motor control. The orientation of the motor rotor can be determined from a feedback source mounted to the motor. The actual commutation source is the motor feedback device assigned to Feedback 1. Once the feedback device is calibrated to the absolute orientation of the rotor using the Commutation Offset attribute, the commutation block can then generate the true Electrical Angle of the rotor. This signal is used to perform the vector transforms between the rotary and stationary motor frames and can also be used for any other algorithms that require knowledge of rotor position.

**Figure 5-46.42 Closed Loop Current Vector Control**



## 5-46.49 Definition of Motion Terms

**Axis** - logical element of a motion control system that exhibits some form of movement. Axes can be rotary or linear, physical or virtual, controlled or simply observed. A physical axis may include one or more of the following components, a motion sensor, a motion control structure, a power amplifier, and a motion actuator.

**Motion** – refers to any aspect the dynamics of an axis. In the context of this document it is not limited to servo drives but encompasses all forms of drive based motor control.

**Drive** – a device designed to control the dynamics of a motor.

**CIP Motion** – defines extensions to CIP objects and device profiles to support motion control over CIP networks.

**CIP Sync** – defines extensions to CIP objects and device profiles to support time synchronization over CIP Networks.

**CIP Motion I/O Connection** – name given to the periodic bi-directional, Class 1, CIP connection between a controller and a drive that is defined as part of the CIP Motion standard.

**CIP Motion Peer Connection** – name given to the periodic multicast, producer/consumer, CIP connection between peer devices in a motion control system that is defined as part of the CIP Motion standard.

**CIP Motion Drive** – refers to any drive that complies with the CIP Motion standard.

**Variable Frequency Drive (VFD)** – name given to a class of drive products that seek to control the speed of a motor, typically an induction motor, through a proportional relationship between drive output voltage and commanded output frequency. Frequency drives are, therefore, sometimes referred to as a Volts/Hertz drives.

**Vector Drive** – name given to a class of drive products that seek to control the dynamics of a motor via closed loop control which includes, but is not limited to, closed loop control of both torque and flux vector components of motor stator current relative to the rotor flux vector.

**Closed Loop** – refers to methods of control where there is a feedback signal of some kind that is used to drive the actual dynamics of the motor to match the commanded dynamics by servo action. In most cases there is a literal feedback device to provide this signal, but in some cases the signal is derived from the motor excitation (i.e. sensorless operation).

**Open Loop** – refers to methods of control where there is no application of feedback to force the actual motor dynamics to match the commanded dynamics. Examples of open loop control are stepper drives and variable frequency drives.

**CIP Motion Drive Profile** – collection of objects used to implement a CIP Motion Drive device that includes the Motion Axis Object, as well as standard support objects like the Identity Object and the CIP Sync Object.

**Motion Axis Object** – describes an object that defines the attributes, services, and behavior of a motion device based axis according to the CIP Motion standard.

**Motion Control Axis Object** – describes an object that defines the attributes, services, and behavior of a controller based axis according to the CIP Motion standard.

**System Time** – absolute time value as defined in CIP Sync standard in the context of a distributed time system where all devices have a local clock that is synchronized with a common master clock. In the context of CIP Motion, System Time is a 64-bit integer value in units of nanoseconds with a value of 0 corresponding to January 1, 1970.

**Set-point – in the context of a device**, a value sent to the drive that is used to directly control some aspect of the motor dynamics, which includes (but not limited to) position, velocity, acceleration, and torque.

**Inverter** – a device that generally converts DC input to AC output. An Inverter is also commonly called the Drive Amplifier. In the context of a drive system, the Inverter is responsible for controlling the application of DC Bus power to an AC motor.

**Converter** – a device that generally converts AC input to DC output. A Converter is also commonly called the Drive Power Supply. In the context of a drive system, the Converter is responsible for converting AC Main input into DC Bus power.

**Shunt Regulator** – refers to a specific Bus Regulator method that switches the DC Bus across a power dissipating resistor to dissipate the regenerative power of a decelerating motor.

**Bus Regulator** – refers to any method used to limit the rise in DC Bus voltage level that occurs when decelerating a motor.

**Thermal Model** – refers to any algorithm that attempts to model the thermal behavior of the associated device during operation.

**Get/Read** – operation that involves the retrieving an attribute value from the perspective of Controller side of the interface.

**Set/Write** – operation that involves the setting an attribute to a specified value from the perspective of Controller side of the interface.

**Cyclic Data Block** – refers to high priority real-time data block that is transferred by a CIP Motion connection on a periodic basis.

**Event Data Block** – refers to medium priority real-time data block that is transferred by a CIP Motion connection only after a specified event occurs. Registration and marker input transitions are typical drive events.

**Service Data Block** - lower priority real-time data block associated with a service message from the controller that is transferred by a CIP Motion connection on a periodic basis. Service data includes service request messages to access Motion Axis Object attributes or perform various drive diagnostics.

**Synchronized** - condition where the local clock value on the drive is locked onto the master clock of the distributed System Time. When synchronized, the drive and controller devices may utilize time stamps associated with CIP Motion connection data.

**Time Stamp** – refers to System Time Stamp value associated with the CIP Motion connection data that conveys the absolute time when the associated data was captured or can be also used to determine when associated data is to be applied.

**Time Offset** – refers to System Time Offset value associated with the CIP Motion connection data that is associated with source device. The System Time Offset is a 64-bit offset value that is added to a device's local clock to generate System Time for that device.

## 5-47 Time Sync Object

**Class Code:** 43<sub>hex</sub>

### 5-47.1 CIP Sync Overview

CIP Sync is the name given to the time synchronization technology for the Common Industrial Protocol (CIP). This technology will allow accurate real-time synchronization of devices and controllers connected over CIP networks that require time stamping, sequence of events recording, distributed motion control, and support for highly distributed applications requiring increased control coordination.

CIP Sync is based on the IEEE 1588 (IEC 61588) standard – Precision Clock Synchronization Protocol for Networked Measurement and Control Systems, commonly referred to as the Precision Time Protocol (PTP). The standard is designed for, but not limited to, local area networks like Ethernet. The protocol provides a standard mechanism to synchronize clocks across a network of distributed devices.

CIP Sync defines the Time Sync Object. The Time Sync Object provides a CIP interface to the IEEE 1588 (IEC 61588) Standard.

### 5-47.2 Definitions

**Boundary clock:** A boundary clock is a clock with more than a single PTP port, with each PTP port providing access to a separate PTP communication path. Boundary clocks are used to eliminate fluctuations produced by routers and similar network elements.

**Clock:** A device providing a measurement of the passage of time since a defined epoch. There are two types of clocks in 1588: boundary clocks and ordinary clocks.

**Epoch:** The reference time defining the origin of a time scale is termed the epoch.

**Grandmaster clock:** Within a collection of 1588 clocks one clock, the grandmaster clock, will serve as the primary source of time to which all others are ultimately synchronized.

**Master clock:** A system of 1588 clocks may be segmented into regions separated by boundary clocks. Within each region there will be a single clock, the master clock, serving as the primary source of time. These master clocks will in turn synchronize to other master clocks and ultimately to the grandmaster clock.

**Ordinary clock:** An ordinary clock is a 1588 clock with a single PTP port.

**PTP:** PTP is an acronym for Precision Time Protocol, the name used in the standard for the protocol.

**PTP message:** There are five designated messages types defined by 1588: Sync, Delay\_Req, Follow-up, Delay\_Resp, and Management.

**PTP port:** A PTP port is the logical access point for 1588 communications to the clock containing the port.

**Synchronized clocks:** Two clocks are synchronized to a specified uncertainty if they have the same epoch and measurements of any time interval by both clocks differ by no more than the specified uncertainty. The timestamps generated by two synchronized clocks for the same event will differ by no more than the specified uncertainty.

**Time Sync Object, Class Code: 43<sub>hex</sub>**

**System Time** – absolute time value as defined by CIP Sync in the context of a distributed time system where all devices have a local clock that is synchronized with a common master clock. System Time is a 64-bit integer value in units of nanoseconds or microseconds with a value of 0 corresponding to an epoch of January 1, 1970.

#### **5-47.3**

#### **Overview of the Precision Time Protocol (PTP)**

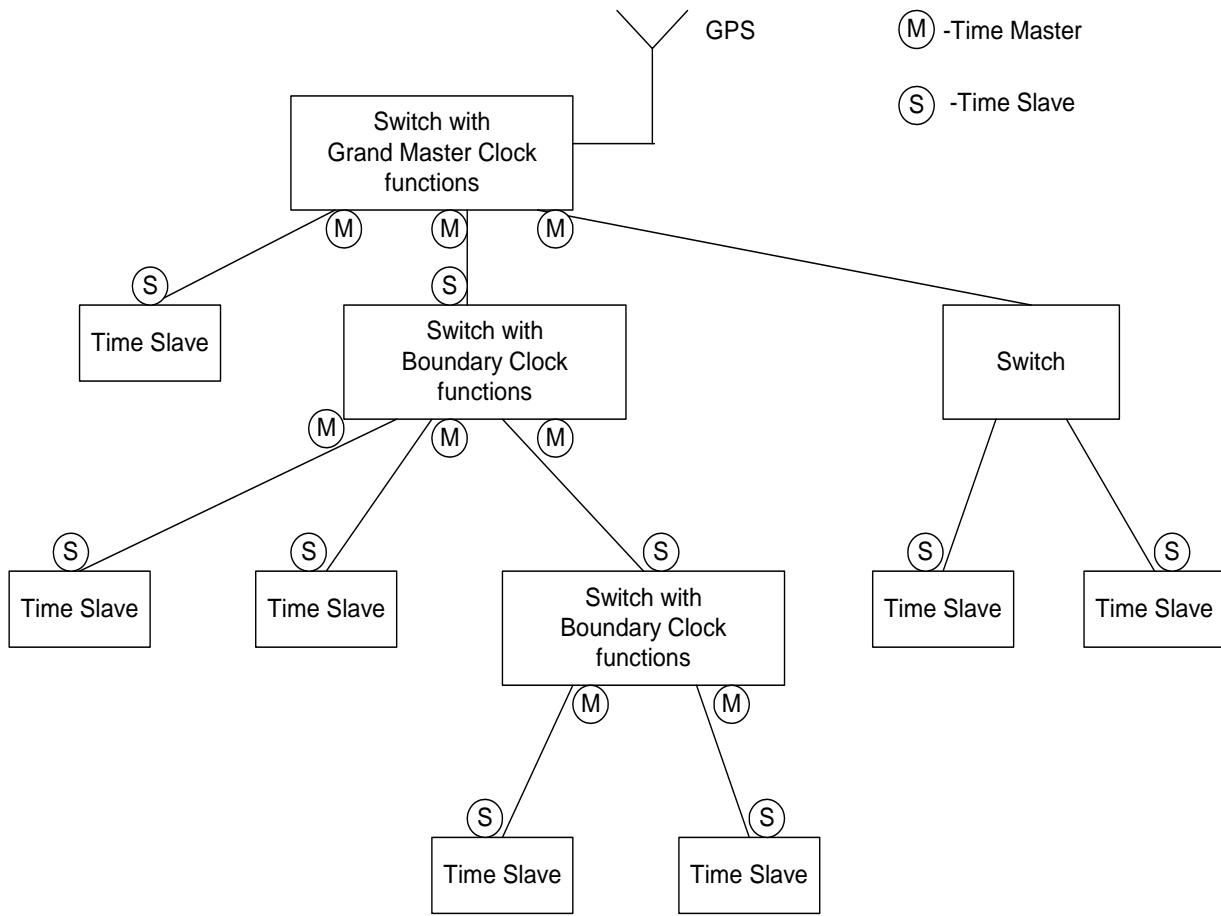
From the 1588 specification: “The IEEE 1588 standard specifies a protocol to synchronize independent clocks running on separate nodes of a distributed measurement and control system to a high degree of accuracy and precision. The clocks communicate with each other over a communication network. In its basic form, the protocol is intended to be administration free. The protocol generates a master slave relationship among the clocks in the system. Within a given subnet of a network there will be a single master clock. All clocks ultimately derive their time from a clock known as the grandmaster clock. The communication path between any clock and its grandmaster clock is part of a minimum spanning tree.”

A PTP system of distributed clocks consists of ordinary clocks and boundary clocks. A boundary clock is a clock with a clock port for each of two or more distinct communication paths. For example a switch that implements the PTP protocol on each of its ports is a boundary clock.

One clock on each subnet in the system is selected as the master clock. The selection of a master is made by each of the other clocks by examining information contained in special timing messages called Sync messages. A sync message is sent periodically by any port claiming to be the master clock. All ports use the same algorithm, termed the Best Master Clock Algorithm (BMCA). If a port of a master clock receives a Sync message from a better clock then that port will cease to claim to be a master and the receiving port will assume the status of a slave. Likewise if a clock with a port acting as a slave determines that it would make a better master than the current master clock or if there is no current master clock, it assumes the status of master and begins to send Sync messages. Some nodes may be implemented as slave only and will never assume mastership (e.g. an I/O device).

An example configuration is pictured in Figure 5-47.1. A system will typically consist of one or more devices capable of becoming a master clock (i.e. controllers, standalone and integrated GPS or NTP time keepers), slave only clocks (i.e. I/O), and other devices that may or may not support time synchronization (switches, I/O, etc.).

Figure 5-47.1 Example System with Grandmaster, Boundary, and Slave Clocks



#### 5-47.4 CIP Sync System Time Definition

The starting date/time for CIP Sync System Time is 12:00 a.m., January 1, 1970. This is the starting epoch for both PTP and System Time. System Time is represented as a 64-bit value (LINT) in nanoseconds or microseconds. System Time is adjusted for leap seconds, but not local time zones or daylight savings time. It represents the current time at the 0th meridian. A negative System Time represents a time prior to the starting date/time.

In the Precision Time Protocol, time is distributed as a structure of type “TimeRepresentation”. This structure is made up of two 32-bit members. The first member of the structure represents the number of seconds since the beginning of the epoch (January 1, 1970 0::00::00). The second member represents the number of nanoseconds. Leap seconds are distributed by the Precision Time Protocol as the current\_utc\_offset field.

PTP time and leap seconds are converted to microseconds or nanoseconds to give System Time as:

$$\text{SystemTime} = \text{PTPTime} - \text{LeapSeconds}$$

#### 5-47.5 CIP Sync Implementation

Each node on the CIP network interested in clock synchronization will contain an implementation of a PTP clock as well as an implementation of the PTP software protocol stack.

#### **5-47.5.1 PTP Clock Implementation**

A PTP clock can be implemented using a hardware assist circuit or with software using a free running timer. The choice of whether to implement a hardware assist clock or a software clock will depend on cost and accuracy considerations. The hardware assist circuit will generally result in a more accurate clock usually in the sub-microsecond range while a software implemented clock will yield accuracy in the tens of microseconds range. The 1588 specification does not describe the clock implementation, but does contain guidelines and requirements.

#### **5-47.5.2 PTP Protocol Stack Implementation**

The PTP protocol stack implementation contains the software that implements the 1588 PTP protocol. This includes the messaging between clocks to synchronize slave clocks to master clocks, the algorithms to process message timestamps and tune the clock, and the management message support required for clock administration. The implementation of the basic protocol is straightforward.

### **5-47.6 CIP Sync Clock Model**

CIP Sync defines an offset clock model to address the requirements for industrial control applications. This model is needed because, even though the 1588 PTP protocol defines a mechanism for distributing and synchronizing time, it does not define a mechanism to compensate for step changes in time that may occur at the grandmaster clock source. These changes may occur due to one or more of the following conditions:

1. The user adjusts the master clock whose type is “HAND” set.
2. A master with a more accurate clock becomes available (new grandmaster). This may occur during system startup or after the system has been running for some time.
3. The time master is temporarily disconnected from the slave clock and then re-connected. In this situation, given any discrepancy in time between the master and the slave, a step change will occur.

Those applications requiring a stable clock in the presence of step changes in time should implement their PTP clock as a local clock, synchronized to the PTP master frequency, but not to the PTP master’s absolute time value.

In this model, the PTP protocol is used to discipline the local clock so that it ticks at the same rate as the master. The device then maintains an offset between the local clock time and system time. A small delta change in time will cause the device to make a small adjustment to the offset and to continue to “tune” its clock. A large change in time will cause the device to update its offset but not “tune” its clock.

Cyclic events may then be scheduled based on the local clock and will not be affected by large step changes in time -- provided that the local clock remains synchronized to the PTP master clock frequency.

Figure 5-47.2 shows the relationship between the Local Clock, System Time Offset, and the System Time for a PTP Time Slave. The “system to local clock offset” is a memory variable maintained by the PTP clock implementation to make offset adjustments. A leap second value is added to the offset to give System Time in terms of the proper epoch.

Figure 5-47.2 CIP Sync Offset Clock Model

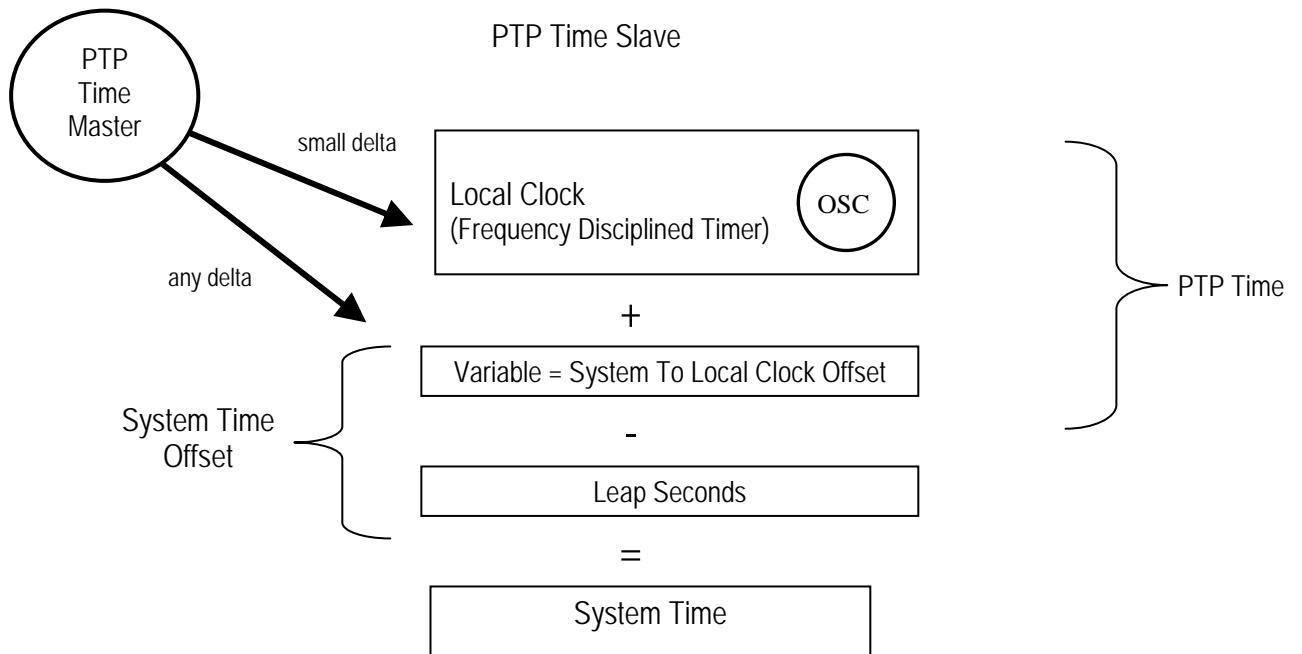
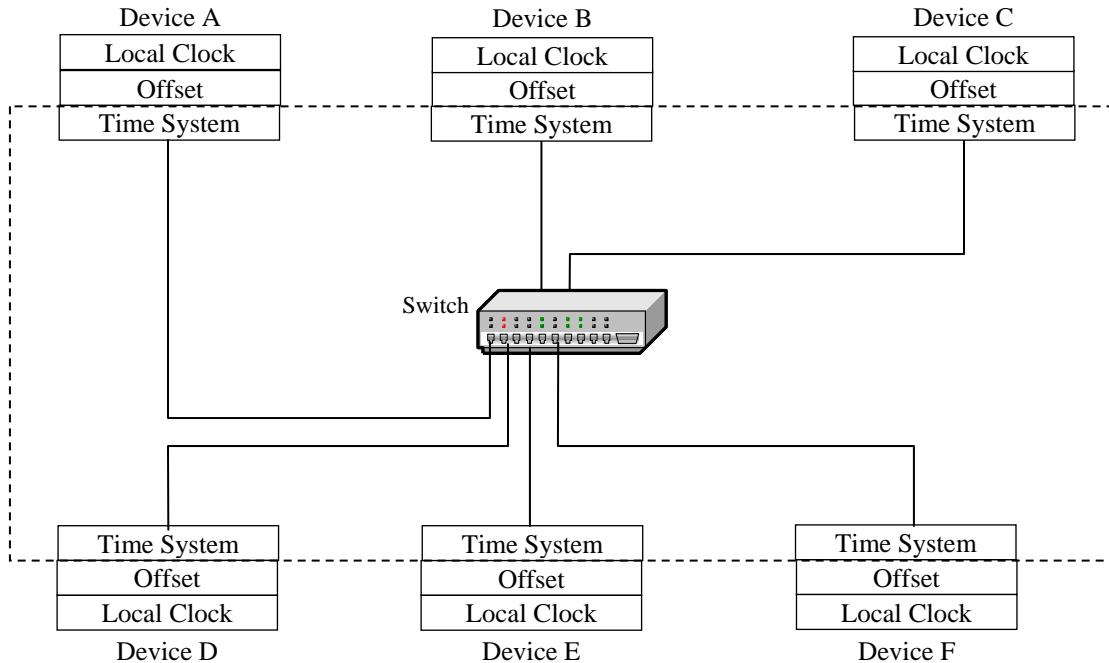


Figure 5-47.3 shows a system of devices each implementing the CIP Sync Offset Clock Model. Each device is a part of a network of devices that share the same concept of System Time. Each device also has a Local Clock value based on a frequency-disciplined timer and related to System Time via an offset value (System Time Offset). Such a system allows each device to maintain a local time independent from all other devices, but share a common notion of system time. As such, System Time may change without requiring changes to the local clock.

Figure 5-47.3 CIP Sync System with Offset Clock Model



**5-47.7 CIP Sync System Time Compensation**

An application creates a timestamp by reading the clock that has been synchronized by PTP and making any required adjustments to the value, for example, converting it to System Time. Additional adjustments may be required if a step has occurred in System Time.

CIP Sync defines a mechanism to maintain a consistent set of timestamps in the presence of step changes to System Time. A step change in System Time effectively causes a shift in the time base of the system. Any step changes to the grandmaster clock time must propagate through the system. Since, all nodes in the system will not see the step change at the same time, a timestamp taken on one node may be inconsistent with a timestamp on another node. Or a timestamp taken at one instance in time may be inconsistent with a timestamp taken later in time on the same node. A compensation algorithm is needed to make timestamps consistent with each other before they are used in computations.

Two possible algorithms are described in the following section.

The decision to compensate timestamps is made based on the requirements of the application.

**5-47.7.1 Step Compensation Algorithms**

An offset value is maintained with each timestamp. This offset value is the System Time Offset at the time the timestamp is captured. The offset can be used to provide an indication of when a step occurs in System Time. If the timestamp is transmitted across the network from one node to a second node the offset is sent along with the timestamp. If the two timestamps are captured within the same node, the offsets are stored along with the timestamps.

The algorithms transform a timestamp from one time base to a second time base if a time base change occurs between the time the two timestamps are captured. When the two timestamps are compared they will reference the same time base. These algorithms may equally be applied to a set of timestamps.

**5-47.7.1.1 Compensation Algorithm for Timestamps Within a Single Node**

This algorithm is defined to allow a node to adjust the value of one or more timestamps that have been captured on the local node.

The algorithm is stated as follows:

$$\text{Timestamp}_C = \text{Timestamp}_0 + \text{Offset}_1 - \text{Offset}_0$$

Where,

$\text{Timestamp}_C$  is the compensated timestamp

$\text{Timestamp}_0$  is the timestamp to be compensated

$\text{Offset}_1$  is the offset associated with the timestamp at the target time base

$\text{Offset}_0$  is the offset for the timestamp to be compensated

The example in the following table compares two timestamps T1 and T2. At time T2 a step of 1000 occurs. The timestamp at T1 is adjusted to 1200 to be consistent with T2 for comparison.

Figure 5-47.4 Example: Compensated Timestamp on a Single Node

The diagram shows a table with columns: Index, Timestamp, and Offset. The rows represent indices T0, T1, T2, and T3. The timestamp values are 100, 200, 1300, and 1400 respectively. The offset values are 0, 0, 1000, and 1000 respectively. A callout box labeled "Step" points to the row at index T2. The box contains the text: "Compensate Timestamp at time T1", "Timestamp<sub>(T1)</sub> = Timestamp<sub>(T1)</sub> + Offset<sub>(T2)</sub> - Offset<sub>(T1)</sub>", and "Timestamp<sub>(T1)</sub> = 200 + 1000 - 0 = 1200".

Index	Timestamp	Offset
T0	100	0
T1	200	0
T2	1300	1000
T3	1400	1000

### 5-47.7.1.2 Compensation Algorithm for Timestamps between Nodes

This algorithm is defined to allow a node to adjust the value of a received timestamp from a remote source node so that it can be compared to a timestamp on its own node, the destination node.

The destination node must be able to detect a step change to the System Time on the remote node or on its own node and make the appropriate adjustment. Two conditions are possible:

1. The source device has seen a step change in time but the destination device has not or
2. The destination device has seen a step change in time but the source device has not.

The step change is detected by a change in the SystemTimeOffset by either the source or destination. The source offset is sent to the destination device along with the timestamp. The destination device compares the offset received to the previously received offset to determine if a step change has occurred and adjusts the received timestamp value accordingly.

The algorithm is stated as follows:

$$\text{Timestamp}_{\text{Compensated}} = \text{Timestamp}_{\text{Received}} + ((\text{Dest}_{\text{Offset}} - \text{Dest}_{\text{LastOffset}}) - (\text{Source}_{\text{Offset}} - \text{Source}_{\text{LastOffset}}))$$

Where:

$\text{Timestamp}_{\text{Received}}$  is received timestamp.

$\text{Dest}_{\text{Offset}}$  is the current value of the local clock time offset at the destination.

$\text{Dest}_{\text{LastOffset}}$  is the previous value of the local clock time offset at the destination.

$\text{Source}_{\text{Offset}}$  is the received value of the local clock time offset from the source.

$\text{Source}_{\text{LastOffset}}$  is the previous value of the local clock time offset from the source.

Last offsets are updated when:

$$|(\text{Dest}_{\text{Offset}} - \text{Dest}_{\text{LastOffset}}) - (\text{Source}_{\text{Offset}} - \text{Source}_{\text{LastOffset}})| \leq \text{SyncBoundsLimit}$$

Where:

$\text{SyncBoundsLimit}$  is a relatively small number that defines the synchronization bounds for the application.

The example in the following table compares two timestamps between nodes after a system time step of 1000. For illustration purposes, a packet is assumed to be sent at the sync interval with a propagation delay of 0; the offsets are arbitrarily assumed to be the same and initially 0, and the syncBoundsLimit is assumed to be a number smaller than 1000.

Figure 5-47.5 Example: Compensated Timestamp Between Nodes

The diagram illustrates a step change in the master clock at interval T3. A callout labeled "Step" points to the master time column for T3, which shows a value of 200 + 1000. An arrow points from this value to the "Adjusted Time" box, which contains the value 200. This indicates that the timestamp received at the destination node is compensated by the step change (1000) and adjusted to a value consistent with the destination clock.

Interval	Master Time	Source Node					Destination Node				
		Local Clock	Offset	Last Offset	Sys Time	Sent Time stamp	Local Clock	Offset	Last Offset	Sys Time	Recv Time stamp
T1	0	0	0	0	0	0	0	0	0	0	0
T2	100	100	0	0	100	100	100	0	0	100	100
T3	200 + 1000	200	0 + 1000	0	1200	1200	200	0	0	200	200
T4	1300	300	1000	0	1300	1300	300	0 + 1000	0	1300	1300
T5	1400	400	1000	1000	1400	1400	400	1000	1000	1400	1400

At interval T3 a step change of 1000 occurs in the master clock and is seen by the source node clock. The timestamp received at the destination clock will be compensated for by 1000 and adjusted to a value consistent with the destination clock which has not yet seen the step change (recv timestamp adjusted from 1200 to 200). At interval T4 the destination node sees the step change but no adjustment is made to the timestamp because the source and destination compensations cancel out. After interval T4 and shown in interval T5, the last offset values are updated.

A comparison of the compensated received timestamp and the current time system time will yield a meaningful and consistent result (200 and 200 vs. 1200 and 200).

## 5-47.8 CIP Sync Group Startup Sequence

CIP Sync defines a group startup sequence and a group synchronizing service. The group startup sequence allows a group of devices to guarantee that their clocks have all been synchronized to the PTP master reference clock before starting an application that depends on System Time. The application defines the specific requirements needed for the group to be considered synchronized.

Each application (object) that wishes to synchronize to a group will implement the GroupSync service. The GroupSync service is a CIP Common service. See ODVA/CI CIP Common Specification, Volume 1, Appendix A.

An application may send additional parameters as part of the group sync service. For example, it may exchange the SystemTimeOffset attribute to facilitate timestamp compensation, or a period and phase to initiate a synchronized periodic event, or one or more bounding parameters to specify a target synchronization requirement.

The service returns true if the device is synchronized to the PTP Time master otherwise it returns false.

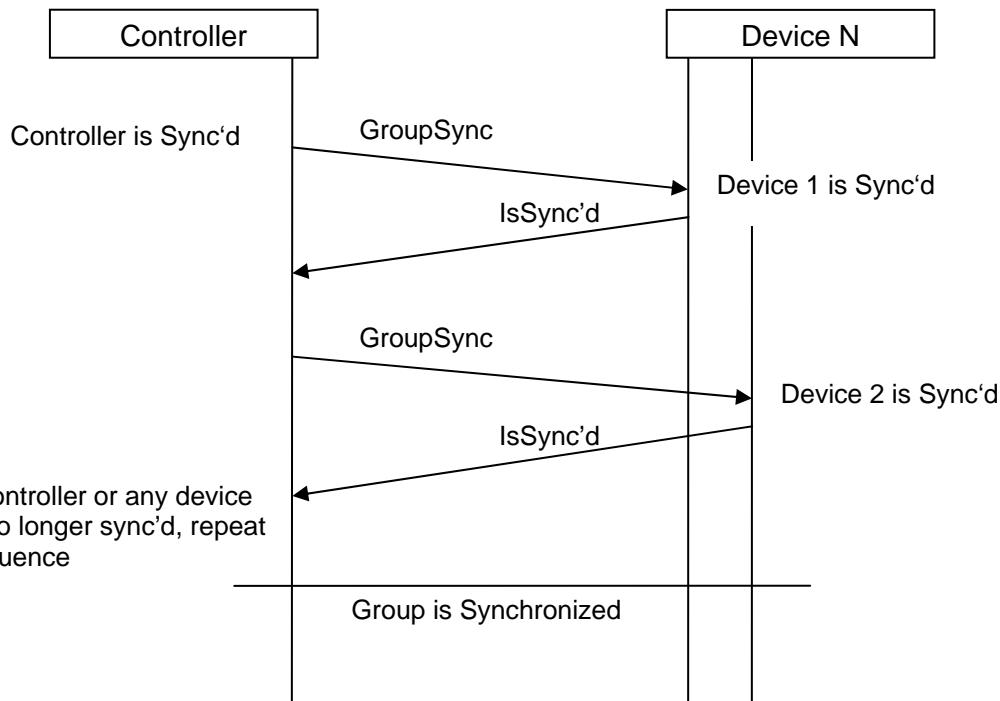
An example startup sequence for a group of applications that need to synchronize is illustrated in the figure below.

Time Sync Object, Class Code: 43<sub>hex</sub>

Once the controller itself is synchronized to the master time clock, it initiates a series of GroupSync messages to each of the devices also part of the same group. Each device returns a response indicating whether it is also synchronized with the PTP master reference clock. If the controller and all devices are synchronized then the group is synchronized. Otherwise, the master application repeats the synchronizing sequence or takes some fault action.

The Sync'd status of each node is determined by the requirements of the application and may be contingent on additional synchronization parameters.

**Figure 5-47.4 Example Startup Sequence**



### 5-47.9 CIP Sync Quality of Service (QoS)

On EtherNet/IP, CIP Sync shall utilize the standard EtherNet/IP frame prioritization scheme to be published in Volume 2. This scheme has not yet been finalized but it is likely to utilize Tagged Frames, as defined by IEEE 802.1p, which has been formally incorporated into 802.1Q. Some CIP Sync applications may require the use of Tagged Frames to achieve the required level of performance. Therefore, designers must consider the selection of hardware and firmware drivers to support "Tagged Frames" with user-selectable priority in their design.

**Table 5-47.1 QoS vs CIP Network Adaptation**

CIP Network	QoS
EtherNet/IP	802.1Q tagged frame
DeviceNet	N/A
CompoNet	N/A
ControlNet	scheduled priority

### 5-47.10 Time Sync Status Visualization

A device shall provide visual indication of clock synchronization status. Table 5-47.2 specifies the visual status of synchronization for various status indicators.

A device is considered synchronized when its local clock is synchronized with the master reference clock. However, an application may impose more stringent synchronization requirements.

**Table 5-47.2 Time Sync Status Visualization**

Visual Status Indicator	Operation
Module Status LED	Standby (flashing green) = not synchronized Operational (Solid green) = synchronized + other device requirements
Seven Segment Display	A device waiting for connection to the controller will display “0” A device configuring will display “1” A device waiting for synchronization will display “2”
Multi-character Alpha-Numeric Display	A device waiting for synchronization will display “Synching” A device that is a PTP Master will display “Time Master” A device that is a PTP Grandmaster will display “Time GM”
Web or Property Page (suggested)	Identity of the current Grandmaster clock Identity of the parent clock of a device Current system time Current grandmaster identifier (e.g. GPS, NTP, HAND, ATOM) Current grandmaster stratum Current offset to master Current observed variance of parent clock Current port status

### 5-47.11 Time Sync Object Overview

The Time Sync Object provides a CIP interface to the IEEE 1588 (IEC 61588) Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems, commonly referred to as the Precision Time Protocol (PTP). Any device supporting CIP Sync shall provide a single instance (Instance 1) of the Time Sync Object. Reference the IEEE 1588 (IEC 61588) Standard for additional details.

The object provides attributes and services to:

1. Get clock status and properties such as synchronized state, current offset to master, and grandmaster UUID.
2. Access PTP clock management functions on an individual clock, such as clock initialization, preferred mastership, and sync interval.

Access the PTP network of devices via native PTP management messages.

### 5-47.12 Class Attributes

The Time Sync Object shall support the following class attributes.

**Table 5-47.3 Time Sync Object Class attributes**

Attrib ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of values
1 thru 7	These class attributes are optional and are described in Volume 1, Chapter 4 of the CIP Networks Library.					

### 5-47.13 Instance Attributes

The Time Sync Object supports the following instance attributes.

**Table 5-47.4 Time Sync Object Instance Attributes**

Attr ID	Need in Implem	Access Rule	Name	Data Type	Description of Attribute	Semantics of values
1	Required	Set	EnablePTP	BOOL	Enable PTP	See “Semantics” section
2	Required	Get	IsSynchronized	BOOL	Local clock is synchronized with reference clock	See “Semantics” section
3	Required	Set	CurrentTimeMicroseconds	LINT	Current value of local_time in microseconds	See “Semantics” section
4	Required	Set	CurrentTimeNanoseconds	LINT	Current value of local_time in nanoseconds	See “Semantics” section
5	Required	Get	OffsetToMaster	LINT	Offset between local clock and master clock	See “Semantics” section
6	Required	Set	MaxOffsetToMaster	LINT	Maximum offset between local clock and master clock	See “Semantics” section
7	Required	Get	DelayToMaster	LINT	Path delay to master	See “Semantics” section
8	Required	Get	GrandMasterClockInfo	STRUCT of	Grandmaster Clock Info	
			Identifier	STRING[4]	Clock Identifier	See “Semantics” section
			Stratum	INT	Clock Stratum	See “Semantics” section
			Variance	INT	Clock Variance	See “Semantics” section
			CommunicationTechnology	INT	Clock communication technology	See “Semantics” section
			PortId	INT	Port Identifier	See “Semantics” section
			UUID	ARRAY of SINT[6]	Clock UUID	See “Semantics” section
9	Required	Get	ParentClockInfo	STRUCT of	Parent Clock Info	
			Reserved	DINT	reserved	
			ObservedDrift	DINT	Observed Drift	See “Semantics” section
			ObservedVariance	INT	Observed Variance	See “Semantics” section
			Variance	INT	Clock Variance	See “Semantics” section
			CommunicationTechnology	INT	Clock communication technology	See “Semantics” section
			PortId	INT	Port Identifier	See “Semantics” section

Time Sync Object, Class Code: 43<sub>hex</sub>

Attr ID	Need in Implem	Access Rule	Name	Data Type	Description of Attribute	Semantics of values
			UUID	ARRAY of SINT[6]	Clock UUID	See “Semantics” section
10	Required	Get	LocalClockInfo	STRUCT of	Local Clock Info	
			Identifier	STRING[4]	Clock Identifier	See “Semantics” section
			Stratum	INT	Clock Stratum	See “Semantics” section
			Variance	INT	Clock Variance	See “Semantics” section
			CommunicationTechnology	INT	Clock communication technology	See “Semantics” section
			PortId	INT	Port Identifier	See “Semantics” section
			UUID	ARRAY of SINT[6]	Clock UUID	See “Semantics” section
11	Required	Get	NumberOfPorts	SINT	Number of ports	See “Semantics” section
12	Required	Get	PortState	STRUCT of	Port state of each port	See “Semantics” section
				INT	Number of ports	
				ARRAY of SINT	Port state of each port	
13	Required	Set	PortEnable	STRUCT of	Port enable status	See “Semantics” section
				INT	Number of ports	
				ARRAY of SINT	Port enable status of each port	
14	Required	Set	PortBurstEnable	STRUCT of	Port burst enable status	See “Semantics” section
				INT	Number of ports	
				ARRAY of SINT	Port burst enable status of each port	
15	Required	Set	SyncInterval	SINT	Sync Interval	See “Semantics” section
16	Required	Set	PreferredMaster	BOOL	Preferred Master	See “Semantics” section
17	Required	Set	Subdomain	STRING[16]	PTP Clock subdomain	See “Semantics” section
18	Required	Set	ClockMode	SINT	Clock Mode	See “Semantics” section
19	Optional	Get	StepsRemoved	INT	Steps Removed	See “Semantics” section
20	Optional	Get	SystemTimeOffset	LINT	System Time Offset	See “Semantics” section

**5-47.13.1 Semantics****5-47.13.1.1 Attribute 1: EnablePTP**

EnablePTP specifies whether the Precision Time Protocol is enabled for this device. The value is 1 if PTP is enabled and 0 if PTP is not enabled.

**5-47.13.1.2 Attribute 2: IsSynchronized**

IsSynchronized specifies whether the local clock is synchronized with the reference clock. The value is 1 if the local clock is synchronized and 0 if the local clock is not synchronized. A clock is synchronized if it has one port in the slave state and is receiving updates from the time master. However, an application may impose more stringent synchronization requirements.

**5-47.13.1.3 Attribute 3: CurrentTimeMicroseconds**

CurrentTimeMicroseconds specifies a 64-bit value of the current time in units of microseconds starting from January 1, 1970 at 12:00 a.m., Universal Coordinated Time (UTC). Leap seconds are distributed by the Precision Time Protocol as the current\_utc\_offset field.

PTP time and leap seconds are converted to microseconds to give the current System Time as:

$$\text{CurrentTime} = \text{PTPTime} + \text{LeapSeconds}$$

Refer to the 1588 specification for a more detailed description of PTP time and the current\_utc\_offset field.

**5-47.13.1.4 Attribute 4: CurrentTimeNanoseconds**

CurrentTimeNanoseconds is the same as CurrentTimeMicroseconds except in units of nanoseconds.

**5-47.13.1.5 Attribute 5: OffsetToMaster**

OffsetToMaster specifies the amount of deviation between the local clock and the reference clock in nanoseconds. Refer to the 1588 specification for more detail. Also refer to the offset\_from\_master data member of the PTP current dataset.

**5-47.13.1.6 Attribute 6: MaxOffsetToMaster**

MaxOffsetToMaster specifies the maximum amount of deviation between the local clock and the reference clock in nanoseconds since last reinitialized. The MaxOffsetToMaster is settable, so that it may be reset.

**5-47.13.1.7 Attribute 7: DelayToMaster**

DelayToMaster specifies the path delay between the local clock and master clock in nanoseconds. Refer to the 1588 specification for more detail. Also refer to the one\_way\_delay data member of the PTP current dataset.

**5-47.13.1.8 Attribute 8: GrandmasterInfo****5-47.13.1.9 Attribute 9: ParentInfo****5-47.13.1.10 Attribute 10: ClockInfo**

GrandmasterInfo, ParentInfo, and ClockInfo specify clock property information for the Grandmaster, Parent and Local PTP clock respectively. The data is extracted from the PTP datasets maintained by the PTP subsystem. The CommunicationTechnology, UUID, and PortID constitute a clocks' Universally Unique Identifier (UUID). The clock identifier, stratum, variance, and drift provide additional information about the properties of the clock.

### 5-47.13.1.10.1 CommunicationTechnology

CommunicationTechnology specifies the communication technology specifier (e.g. Ethernet). Refer to the 1588 specification for more detail. Also refer to the clock\_communication\_technology data member of the PTP default dataset and the parent\_communication\_technology and grandmaster\_communication\_technology of the parent dataset.

**Table 5-47.5 CommunicationTechnology Specifier Meanings**

CIP Network	CommunicationTechnology
EtherNet/IP	1
DeviceNet	7
ControlNet	9
CompoNet	TBD <sup>1</sup>

<sup>1</sup> Value not yet assigned by IEEE

### 5-47.13.1.10.2 UUID

UUID specifies the Universally Unique IDentifier (UUID) of the associated clock. For all CIP Networks except EtherNet/IP, the two-byte vendor id followed by the four byte serial number will be the defined UUID. Refer to the 1588 specification for a more detailed description. Also refer to the clock\_uuid\_field data member of the PTP default dataset and the parent\_uuid\_field and grandmaster\_uuid\_field of the parent dataset.

**Table 5-47.6 UUID Definition for CIP Network Adaptations**

CIP Network	UUID
EtherNet/IP	MAC Address
DeviceNet	Vendor Id / Serial Number
ControlNet	Vendor Id / Serial Number
CompoNet	Vendor Id / Serial Number

### 5-47.13.1.10.3 PortId

PortId specifies the port identifier for the clock. For an ordinary clock the PortId value is 1. Refer to the 1588 specification for more detail. Also refer to the clock\_port\_id\_field data member of the PTP default dataset and the parent\_port\_id\_field and grandmaster\_port\_id\_field of the parent dataset.

### 5-47.13.1.10.4 Identifier

Identifier specifies the expected absolute accuracy and epoch of a given clock (e.g. GPS, NTP, HAN). Refer to the 1588 specification for more detail. Also refer to the clock\_identifier data member of the PTP default dataset and the grandmaster\_identifier of the parent dataset.

### 5-47.13.1.10.5 Stratum

Stratum specifies an additional measure of the quality of a clock. Values are defined from 0 to 255 with stratum 1 being the most closely linked to an authoritative source. (e.g. NTP with stratum 1 or 2). Refer to the 1588 specification for more detail. Also refer to the clock\_stratum data member of the PTP default dataset and the grandmaster\_stratum of the parent dataset.

**5-47.13.1.10.6 Variance**

Variance specifies an estimated measure of the inherit stability properties of a clock. Refer to the 1588 specification for more detail. Also refer to the `clock_variance` data member of the PTP default dataset and the `parent_variance` and `grandmaster_variance` of the parent dataset.

**5-47.13.1.10.7 Observed Variance**

Observed Variance specifies an estimated measure of the parent clock's variance as observed by the slave clock. Refer to the 1588 specification for more detail. Also refer to the `observed_variance` data member of the PTP parent dataset.

**5-47.13.1.10.8 Observed Drift**

Observed Drift specifies an estimated measure of the parent clock's drift as observed by the slave clock. Refer to the 1588 specification for more detail. Also refer to the `observed_drift` data member of the PTP parent dataset.

**5-47.13.1.11 Attribute 11: NumberOfPorts**

`NumberOfPorts` specifies the number of PTP ports on the device. PTP Ordinary clocks have one port. Only a PTP Boundary clock will have more than one port. Refer to the 1588 specification for more detail of a PTP port. Also refer to the `number_of_ports` data member of the PTP default dataset.

**5-47.13.1.12 Attribute 12: Port State**

`PortState` specifies the current status of each PTP port on the device as define by the PTP port state. (e.g. Initializing, Master, Slave, Faulted). The port assignments are device specific. Refer to the 1588 specification for more detail. Also refer to the `port_state` data member of the PTP port dataset.

**5-47.13.1.13 Attribute 13: Port Enable**

`PortEnable` specifies the port enable status of each port on the device. The value is set to 1 if the port is enabled and 0 if the port is disabled. Refer to the 1588 specification for more detail. Also refer to the `port_state` data member of the PTP port dataset.

**5-47.13.1.14 Attribute 14: Port Burst Enable**

`PortBurstEnable` specifies the port burst enable status of each port on the device. The value is set to 1 if the port burst is enabled and 0 if the port burst is disabled. Refer to the 1588 specification for more detail. Also refer to the `burst_enabled` data member of the PTP port dataset.

**5-47.13.1.15 Attribute 15: SyncInterval**

`SyncInterval` specifies the PTP sync interval between successive "Sync" messages issued by a master clocks. The attribute does not take effect until after re-initialization. The default is 2 seconds. The value is the logarithm base 2 of the sync interval. Refer to the 1588 specification for more detail. Also refer to the `sync_interval` data member of the PTP default dataset.

**5-47.13.1.16 Attribute 16: IsPreferred**

IsPreferred specifies the “preferred” status of a master clock. The attribute indicates whether the clock is to be preferred in the selection of the grandmaster clock as part of the Best Master Clock Algorithm (BMCA). The attribute does not take effect until after re-initialization. The value is set to 1 if the clock is preferred and 0 if not. The default is 0. Refer to the 1588 specification for more detail. Also refer to the preferred data member of the PTP default dataset.

**5-47.13.1.17 Attribute 17: Subdomain**

Subdomain specifies the PTP clock subdomain. The default subdomain is “\_DFLT”. The attribute does not take effect until after re-initialization. Refer to the 1588 specification for more detail. Also refer to the subdomain\_name data member of the PTP default dataset.

**5-47.13.1.18 Attribute 18: ClockMode**

ClockMode specifies the mode of the clock as “slave only” or “master capable” if this attribute is settable for a given device. It also reports its PTP clock type as an ordinary or boundary clock. A “master capable” device participates in the PTP Best Master Algorithm (BMCA) and may become a PTP master, while a “slave only” clock will not become a PTP master. The ClockMode is not settable for a boundary clock.

**Table 5-47.7 ClockMode Value Definition**

ClockMode	Value
Slave Only / Ordinary Clock	0
Master Capable / Ordinary Clock	1
Master Capable / Boundary Clock	2

**5-47.13.1.19 Attribute 19: StepsRemoved**

StepsRemoved specifies the number of communication paths traversed between the local clock and the grandmaster clock. Refer to the 1588 specification for more detail. Also refer to steps\_removed data member of the PTP default dataset.

**5-47.13.1.20 Attribute 20: SystemTimeOffset**

SystemTimeOffset specifies the offset to the local clock value. This attribute only applies to a device that implements the CIP Sync clock offset model. The responding device will return the current System Time Offset, otherwise zero.

**5-47.14 Common Services**

The Time Sync Object provides the following common services.

**Table 5-47.8 Common services**

Service Code	Need in Implementation		Service Name	Description of Service
	Class	Instance		
03 <sub>hex</sub>	N/A	Required	Get_Attributes_List	Per Appendix A
04 <sub>hex</sub>	N/A	Required	Set_Attributes_List	Per Appendix A
0E <sub>hex</sub>	N/A	Required	Get_Attribute_Single	Per Appendix A
10 <sub>hex</sub>	N/A	Required	Set_Attribute_Single	Per Appendix A

**5-47.15 Object-Specific Services**

**Time Sync Object, Class Code: 43<sub>hex</sub>**

The Time Sync Object shall support the following object-specific services:

**Table 5-47.9 Object specific services**

Service Code	Need in Implementation		Service Name	Description of Service
	Class	Instance		
4B <sub>hex</sub>	n/a	Required	Initialize	Initialize Time Sync Object
4A <sub>hex</sub>	n/a	Conditional <sup>1</sup>	ManagementMessage	Initiate a PTP management message

Note1: Required for Master Capable devices. Optional for Slave Only devices.

**5-47.15.1 Initialize Service**

The Initialize service causes the PTP Clock to initialize to its power up state. The service is typically invoked after a configuration change to the Time Sync object. An initialize service also causes the PTP clock to re-synchronize to the current PTP master reference clock. Refer to the 1588 specification for more detail.

**5-47.15.2 ManagementMessage Service**

The ManagementMessage service causes the recipient to initiate a native PTP management message to one or more PTP clocks. This service provides CIP access to the complete set of PTP management messages. It also provides a broadcast mechanism not present in CIP and needed for certain operations such as InitializeClock to reinitialize all the PTP clocks in the system. It also provides a mechanism for accessing non-CIP network clocks. The list of supported PTP commands is enumerated in Table 5-47.11. Refer to the 1588 standard for a detailed description of each command and the appropriate request and response parameters.

The service request is received by a particular device. The device then issues a corresponding PTP management message using the native communication mechanism (e.g. PTP/UDP over Ethernet). The device waits for the response if a response is expected for the particular command and then responds with an appropriate CIP response, otherwise a success response is sent immediately. The message will propagate through the entire PTP system until it reaches its targeted destination.

**Table 5-47.10 Request/Response Message Format**

Name	Data Type	Description of Parameter	Semantics of values
Management Message Payload	Command specific	Request and/or Response parameters	See specific Management Message sections in the 1588 standard.

**Table 5-47.11 1588 Management Messages**

Management Message Command	
ObtainIdentity	GetGlobalTimeDataSet
InitializeClock	UpdateGlobalTimeProperties
SetSubDomain	GoToFaultyState
ClearDesignatedPreferredMaster	GetForeignDataSet
SetDesignatedPreferredMaster	SetSyncInterval
GetDefaultDataSet	DisablePort
UpdateDefaultDataSet	EnablePort
GetCurrentDataSet	DisableBurst
GetParentDataSet	EnableBurst
GetPortDataSet	SetTime

## 5-48 Connection Configuration Object

### Class Code: F3<sub>hex</sub>

This object defines an interface used to create, configure and control CIP connections in a device. This specification does not define or constrain the operation of the device's connection management state machine.

NOTE: The CCO object is a device addressable object (not a port addressable object). Refer to Chapter 10 for a discussion about scope of objects.

#### 5-48.1 Class Attributes

**Table 5-48.1 Connection Configuration Object Class Attributes**

Attr ID	Need in Implem	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Value
1	Required	Get	NV	Revision	UINT	Third revision, value = 3	See Chapter 5-48.1.1.1
2	Required	Get	NV	Max Instance	UDINT	Maximum instance number	
3	Required	Get	NV	Num Instances	UDINT	Number of connections currently instantiated	
4	This class attribute is optional and is described in Chapter 4						
5	Conditional <sup>5</sup>	Get	NV	Optional service list	Struct of:	List of optional services utilized in an object class implementation	A list of service codes specifying the optional services implemented in the device for this class.
				Number services	UINT	Number of services in the optional service list	The number of service codes in the list
				Optional services	Array of UINT	List of optional service codes	The optional service codes.
6 and 7	These class attributes are optional and are described in Chapter 4						
8	Required	Get	NV	Format Number	UDINT	See Semantics section	
9	Required	Set	NV	Edit Signature	UDINT	See Semantics section	

Connection Configuration Object, Class Code: F3<sub>hex</sub>

Attr ID	Need in Implem	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Value
10 thru 32	Reserved						
33	Optional	Set <sup>1,2</sup>	NV	Scanner Mode	BOOL	The originator to target packets for connections originated by this device shall reflect the state of this attribute.	0 = idle mode 1 = run mode
34	Optional	Get	V	Scanner Capabilities <sup>4</sup>	WORD	Bit 0, set to Scanner Mode (attribute 33) supported <sup>3</sup> Bit 1, set to Scanner Mode (attribute 33) currently supported Bit 2, Instances may currently be created in Run mode Bit 3, Instances may currently be changed in Run mode Bit 4, Instances may currently be deleted in Run mode Bit 5, Instance may currently be created in Idle mode Bit 6, Instances may currently be changed in Idle mode Bit 7, Instances may currently be deleted in Idle mode Bit 8, Open_Connection/Close_Connection services supported Bit 9, Stop_Connection service supported Bit 10, Get_Member service to read image tables supported Bits 11-15 – reserved and shall be zero	Bits 0 – 10 = 0, No = 1, Yes

Connection Configuration Object, Class Code: F3<sub>hex</sub>

Attr ID	Need in Implem	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Value
---------	----------------	-------------	----	------	-----------	--------------------------	--------------------

1 – A set to attribute 33 shall return a device state conflict (0x10) error if changing the Scanner Mode is not permitted when the set attribute command is received.

2 – The value of attribute 33 may be different than the value last set to attribute 33 if another mechanism (like a key switch) changed the scanner's mode.

3 – Bit 0 indicates if the Scanner Mode attribute has the ability to ever be set using a set attribute service.

4 – If the device in which this object is implemented has some sort of mechanism for changing connections, the state of these bits shall be changed to reflect the present state of the device.

5 – Required if object supports any optional services. Optional if object does not support any optional services.

The additions made to this Object with revision 2 are:

Definition of Connection Status attribute for target-side, peer-to-peer connections

In the Connection Flags Attribute defined bits 4-6 for “T=>O Real time transfer format”. Changed the name of bits 1-3 from “Idle Style” to “O=>T Real time transfer format” and defined the value=3 for “Heartbeat”

Changed the Object Specific service Get\_Status from optional to required, and changed the Set\_Attribute\_Single service from required to optional.

Added Instance attribute 11, Proxy Device ID. Updated the Get\_Attributes\_All response and Set\_Attributes\_all request to include attribute 11.

### 5-48.1.1 Semantics

#### 5-48.1.1.1 Revision – UINT data type

Defines the revision of the Connection Configuration Object Class Definition upon which the implementation is based. As time progressed it is possible for technical updates of the Connection Configuration Object Class Definition to occur for which an indication is desireable. This attribute enables these updates to be tracked inside devices.

The current value assigned to this attribute is three (3). If updates which require an increase in this value are made, then the value will be increased. Support of this attribute is required.

**Table 5-48.2 Revision History**

Revision	Reason for change and description of change from previous revision
0x0001	Initial Definition
0x0002	Added class attributes 33 and 34
0x0003	Modification to make class attribute 5 conditional
0x0004 – 0xFFFF	Reserved by CIP

### 5-48.1.1.2 Format Number

The format value that shall be specified in the format field of each instance of attribute 9.

Values:

- 0 – Single O->T/T->O tables, 16-bit words, 0-based offsets
- 1 – Multiple O->T/T->O tables, 16-bit words, 0-based offsets
- 2 - 99 – Reserved
- 100 - 199 – Vendor Specific
- All other values reserved

### 5-48.1.1.3 Edit Signature

Created and used by configuration software to detect modification to the instance attribute values. For ControlNet this value is initially 0 and is incremented each time a change complete operation is performed. For all other CIP networks this value is initially 0 and is set to a 32 bit CRC each time a change complete operation is performed (the polynomial is  $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x^1+x^0$ ), the CRC seed value shall be 0. The CRC is calculated on the set all attribute data stream for each connection instance (lowest to highest) plus class attribute 1, class attribute 2, class attribute 3 and class attribute 8.

## 5-48.2 Instance Attributes

**Table 5-48.3 Connection Configuration Object Instance Attributes**

Attr ID	Need in Implem	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
1	Required	Get	V	Connection Status	Struct of	This attribute is the connection status.	See Semantics section
				gen_status	USINT	General status	
				reserved	USINT	Reserved	Shall be zero
				ext_status	UINT	Extended status	
2	Required	Set	NV	Connection Flags	WORD	Connection flags	See Semantics
3	Required	Set	NV	Target Device ID	Struct of		See Semantics section
				vendor_id	UINT	Vendor ID	
				product_type	UINT	Device Type	
				product_code	UINT	Product Code	
				major_rev	USINT	Major revision	
				minor_rev	USINT	Minor revision	
4	Conditional	Set	NV	CS Data Index Number	UDINT		See Semantics section

**Connection Configuration Object, Class Code: F3<sub>hex</sub>**

Attr ID	Need in Implem	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
5	Required	Set	NV	Net Connection Parameters	Struct of		
				conn_timeout	USINT	Connection Timeout Multiplier	See Semantics section
				xport_class_and_trig	BYTE	Transport Class and Trigger	See Semantics section
				rpi_OT	UDINT	Originator to Target Requested Packet Interval	See Semantics section
				net_OT	UINT	Originator to Target network connection parameters	See Semantics section
				rpi_TO	UDINT	Target to Originator Requested Packet Interval	See Semantics section
				net_TO	UINT	Target to Originator network connection parameters	See Semantics section
6	Required	Set	NV	Connection Path	Struct of		
				open_path_size	USINT	Open connection path size	See Semantics section
				reserved	USINT	Reserved	
				open connection path	Padded EPATH	Connection path	See Semantics section
7	Required	Set	NV	Config #1 Data	Struct of		See Semantics section
				config_data_size	UINT	Length of config_data in bytes.	
				config_data	Array of octet	Config # 1 data	
8	Required	Set	NV	Connection Name	Struct of		See Semantics section
				name_size	USINT	Number of characters in the connection name.	
				reserved	USINT	Reserved	Shall be zero
				connection_name	STRING2	User-assigned connection name encoded in UNICODE.	
9	Required	Set	NV	I/O Mapping	Struct of		See Semantics section
				format_number	UINT	This number determines the format of this attribute.	The format value shall match the value of class attribute 8.
				mapping_data_size	UINT	Size in bytes of the mapping_data field that follows.	

Connection Configuration Object, Class Code: F3<sub>hex</sub>

Attr ID	Need in Implem	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
				mapping_data	Array of BYTE	I/O mapping information associated with this instance.	
10	Required	Set	NV	Config #2 Data	Struct of		See Semantics section
				config_data_size	UINT	Length of config_data in bytes.	
				config_data	Array of octet	Config # 2 data	
11	Required	Set	NV	Proxy Device ID	Struct of		See Semantics section
				vendor_id	UINT	Vendor ID	
				product_type	UINT	Device Type	
				product_code	UINT	Product Code	
				major_rev	USINT	Major revision	
				minor_rev	USINT	Minor revision	
12	Conditional <sup>1</sup>	Set	NV	Connection Disable	BOOL	Indicates if this instance of the connection configuration object is disabled	0 – This instance of the connection configuration object is enabled. 1 – This instance of the connection configuration object is disabled. When an Open service is received this value shall be set to 0. When a Close or Stop service is received this value shall be set to 1.
13	Conditional <sup>2</sup>			Safety Parameters	See CIP Safety Specification (Volume 5, Chapter 5)		
14	Conditional <sup>2</sup>			Safety Connection Parameter CRC			
15	Conditional <sup>2</sup>			Safety Configuration Instance			
16	Conditional <sup>2</sup>			Safety ID Allocation			
17	Conditional <sup>2</sup>			Safety Target Connection Serial Number			
18	Optional	Set		Net Connection Parameters Attribute Selection	USINT	Selects between attribute 5 and attribute 19	See Semantics section

Connection Configuration Object, Class Code: F3<sub>hex</sub>

Attr ID	Need in Implem	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
19	Conditional <sup>3</sup>	Set		Large Net Connection Parameters	Struct of		
				conn_timeout	USINT	Connection Timeout Multiplier	See Semantics section
				xport_class_and_trigger	BYTE	Transport Class and Trigger	See Semantics section
				rpi_OT	UDINT	Originator to Target Requested Packet Interval	See Semantics section
				net_OT	UDINT	Originator to Target large network connection parameters	See Semantics section
				rpi_TO	UDINT	Target to Originator Requested Packet Interval	See Semantics section
				net_TO	UDINT	Target to Originator large network connection parameters	See Semantics section
20	Conditional <sup>2</sup>			Format Type	See CIP Safety Specification (Volume 5, Chapter 5)		
21	Conditional <sup>2</sup>			Format Status			
22	Conditional <sup>2</sup>			Max Fault Number			

- 1 Attribute 12 shall be supported if conditional services Open\_Connection (4C) and Close\_Connection (4D) are supported. Attribute 12 shall not be supported if conditional services Open\_Connection (4C) and Close\_Connection (4D) are not supported.
- 2 These attributes are not allowed if the device is not a safety device. If the device is a safety device, see Volume 5, CIP Safety.
- 3 Conditional attribute 19 is required if attribute 18 is supported, otherwise conditional attribute 19 is not allowed.

**5-48.2.1 I/O Mapping Formats**

The following tables describe the structure of the mapping\_data field of instance attribute 9 for each of the format numbers.

**Table 5-48.4 Mapping Formats**

Format Number	Mapping_data_size (in bytes)	Name	Data Type	Description
0	4	Single I/O tables	Struct of:	16 bit words, 0 based 16 bit word offsets
		O->T offset	UINT	Offset into O->T image table.
		T->O offset	UINT	Offset into T->O image table.
1	8	Multiple I/O tables	Struct of:	Table selection, 16 bit words, 0 based 16 bit word offsets
		O->T table	UINT	O->T image table selection
		O->T offset	UINT	Offset into O->T image table.
		T->O table	UINT	T->O image table selection
		T->O offset	UINT	Offset into T->O image table.

### 5-48.2.2 Semantics

#### 5-48.2.2.1 Connection Status

Table 5-48.5 Originator Connection Status Values

General Status	Extended Status	Description
0x00	n/a	Connection is open
0x01	0x0000 or greater	See the table of Connection Manager Service Request Error Codes in Volume 1, Chapter 3, in the Error Code Listing subsection
0x02 – 0x26	0x0000 or greater	See General Status codes, Volume 1, Appendix B
0xD0	0x0001	Connection is closed or stopped (via the Close or Stop services)
	0x0002	Connection open is pending
	0x0003	Connection close is pending

Table 5-48.6 Target Connection Status Values

General Status	Extended Status	Description
0x00	0x0000	Connection is not open
0x00	0x0001 or greater	The number of connections to this target
0xD0	0x0001	Connection is closed or stopped (via the Close or Stop services)

#### 5-48.2.2.2 Connection Flags

Table 5-48.7 Connection Flags Attribute Definition

Bit	Meaning
0	Connection 0 = Originator 1 = Target
1 – 3	O->T Real time transfer format 000 – Use 32-bit Run/Program header to indicate idle mode, Volume 1, Section 3-6.1.4. 001 – Use zero data length packet to indicate idle mode. 010 – None. Connection is pure data and is modeless. 011 – Heartbeat. 100 – Reserved. 101 – Safety (See Volume 5) 100 thru 111 – reserved for future use.
4 - 6	T->O Real time transfer format 000 – Use 32-bit Run/Program header to indicate idle mode, Volume 1, Section 3-6.1.4 001 – Use zero data length packet to indicate idle mode. 010 – None. Connection is pure data and is modeless. 011 – Heartbeat. 100 – Reserved. 101 – Safety (See Volume 5) 100 thru 111 – reserved for future use.
7 - 15	Reserved

### **5-48.2.2.3 Target Device ID**

This attribute is the identity of the target device for this connection instance. This identity information is not used for verifying (keying) the online target device, it is used by a configuration tool to locate the correct electronic data sheet for the connection configuration described in this CCO instance. For Target instances this attribute shall be the identity of the device containing this CCO object.

### **5-48.2.2.4 CS Data Index Number**

The CS Data Index Number attribute is required for a device on ControlNet; otherwise this attribute shall not be implemented.

The CS Data Index Number is a value set by the configuration software. This is the connection\_index value returned from the Schedule Object read service (See Volume 4, ControlNet Adaptation of CIP, Chapter 5 for the definition of the Scheduling Object services). This attribute is ignored for target instances.

### **5-48.2.2.5 Net Connection Parameters**

#### **5-48.2.2.5.1 conn\_timeout**

The conn\_timeout is the value used for the connection timeout multiplier field defined in Volume 1, Chapter 3 for a description of the Forward\_Open service request. conn\_timeout is ignored for target instances.

#### **5-48.2.2.5.2 xport\_class\_and\_trig**

The xport\_class\_and\_trig is the value used for the Transport Type/Trigger field defined in Volume 1, Chapter 3 for a description of the Forward\_Open service request. xport\_class and\_trig is ignored for target instances.

#### **5-48.2.2.5.3 rpi\_OT**

The rpi\_OT is the value used for the O\_to\_T RPI field defined in Volume 1, Chapter 3 for a description of the Forward\_Open service request. rpi\_OT is ignored for target instances.

#### **5-48.2.2.5.4 net\_OT**

The net\_OT is the value used for the O\_to\_T Network Connection Parameters field in Volume 1, Chapter 3 for a description of the Forward\_Open service request. Only the size field is used for target instances.

#### **5-48.2.2.5.5 rpi\_TO**

The rpi\_TO is the value used for the T\_to\_O RPI field defined in Volume 1, Chapter 3 for a description of the Forward\_Open service request. rpi\_TO is ignored for target instances.

#### **5-48.2.2.5.6 net\_TO**

The net\_TO is the value used for the T\_to\_O Network Connection Parameters field in Volume 1, Chapter 3 for a description of the Forward\_Open service request. Only the size field is used for target instances.

#### **5-48.2.2.6 Connection Path**

##### **5-48.2.2.6.1 open\_path\_size**

The open\_path\_size is the value used for the Connection\_Path\_size field in Volume 1, Chapter 3 for a description of the Forward\_Open service request. For target instances the open\_path\_size is used for matching the open\_path\_size received in a forward open.

##### **5-48.2.2.6.2 open connection path**

The open\_connection\_path is the value used for the Connection\_Path field in Volume 1, Chapter 3 for a description of the Forward\_Open service request. For target instances the open\_connection\_path is used for matching the open\_connection\_path received in a forward open.

#### **5-48.2.2.7 Config #1 Data**

This does not apply to target instances. The data specified in attributes 7 and 10 are concatenated (in that order) into a single data segment, then appended to the connection path in attribute 6 to form the complete path sent in the Forward\_Open service of the Connection Manager object.

All the configuration data may be placed in attribute 7 or all the configuration data may be placed in attribute 10 or the configuration data may be split between attributes 7 and 10. When a connection is a proxied connection the convention that shall be followed is that any configuration data intended for the proxying device is placed in attribute 7 and any configuration data intended for the proxied device is placed in attribute 10. For connections to devices that are configured using the target device EDS this information comes from the Connection Manager section of the EDS, fields 9 and 10 of the Connection entry (see the Connection Manager Section of Volume 1, Chapter 7 for more information). Config #1 data is ignored for target instances.

#### **5-48.2.2.8 Connection Name**

This Connection Name field allows a user to name each connection instance. The connection name has no meaning to the Connection Configuration Object.

#### **5-48.2.2.9 I/O Mapping Attribute**

The I/O Mapping attribute contains I/O mapping data. The I/O mapping data specifies image table locations where target to originator data is placed and where originator to target data is obtained.

#### **5-48.2.2.10 Config #2 Data**

This note does not apply to target instances. The data specified in attributes 7 and 10 are concatenated (in that order) into a single data segment, then appended to the connection path in attribute 6 to form the complete path sent in the Forward Open service of the Connection Manager object. All the configuration data may be placed in attribute 7 or all the configuration data may be placed in attribute 10 or the configuration data may be split between attributes 7 and 10. When a connection is a proxied connection the convention that shall be followed is that any configuration data intended for the proxying device is placed in attribute 7 and any configuration data intended for the proxied device is placed in attribute 10. For connections to devices that are configured using the target device EDS this information comes from the Connection Manager section of the EDS, fields 11 and 12 of the Connection entry (see the Connection Manager Section of Volume 1, Chapter 7 for more information). Config #2 data is ignored for target instances.

### **5-48.2.2.11 Proxy Device ID**

This attribute is the identity of the device proxying for the target device for this connection instance. This identity information is not used for verifying (keying) the online target device, it is used by a configuration tool to locate the correct electronic data sheet for the connection configuration described in this CCO instance. For Target instances and Connections that are not proxied this attribute shall be ignored.

### **5-48.2.2.12 Net Connection Parameters Attribute Selection**

The Net Connection Parameters Attribute Selection attribute selects between the use of instance attribute 5 (Net Connection Parameters) and attribute 19 (Large Net Connection Parameters). The valid values are:

- 0 = indicates instance attribute 5 shall be used (default)
- 1 = indicates instance attribute 19 shall be used
- 2-255 = reserved

If this optional attribute is not supported attribute 5 shall be used.

### **5-48.2.2.13 Large Net Connection Parameters**

#### **5-48.2.2.13.1 conn\_timeout**

The conn\_timeout is the value used for the connection timeout multiplier field defined in Volume 1, Chapter 3 for a description of the Large\_Forward\_Open service request. conn\_timeout is ignored for target instances.

#### **5-48.2.2.13.2 xport\_class\_and\_trig**

The xport\_class\_and\_trig is the value used for the Transport Type/Trigger field defined in Volume 1, Chapter 3 for a description of the Large\_Forward\_Open service request. xport\_class and\_trig is ignored for target instances.

#### **5-48.2.2.13.3 rpi\_OT**

The rpi\_OT is the value used for the O\_to\_T RPI field defined in Volume 1, Chapter 3 for a description of the Large\_Forward\_Open service request. rpi\_OT is ignored for target instances.

#### **5-48.2.2.13.4 net\_OT**

The net\_OT is the value used for the O\_to\_T Network Connection Parameters field in Volume 1, Chapter 3 for a description of the Large\_Forward\_Open service request. Only the size field is used for target instances.

#### **5-48.2.2.13.5 rpi\_TO**

The rpi\_TO is the value used for the T\_to\_O RPI field defined in Volume 1, Chapter 3 for a description of the Large\_Forward\_Open service request. rpi\_TO is ignored for target instances.

#### **5-48.2.2.13.6 net\_TO**

The net\_TO is the value used for the T\_to\_O Network Connection Parameters field in Volume 1, Chapter 3 for a description of the Large\_Forward\_Open service request. Only the size field is used for target instances.

### 5-48.3 Connection Configuration Object Change Control

Changes to Connection Configuration object attributes can only be made after a Change Start service has been successfully issued. Changes to Connection Configuration object attributes are applied after a Change Complete service has been successfully issued.

The following services are valid during a change operation. A change operation is started by issuing a Change Start service and completed by issuing a Change Complete service:

**Table 5-48.8 Services Valid During a Change Operation**

Service Code	Service Name
02hex	Set_Attribute_All
08hex	Create
09hex	Delete
10hex	Set_Attribute_Single
15hex	Restore
4Bhex	Kick Timer
51hex	Change Complete
52hex	Audit Changes

### 5-48.4 Common Services

The Connection Configuration Object provides the following Common Services:

**Table 5-48.9 Connection Configuration Object Common Services**

Service Code	Need in Implementation		Service Name	Description of Service
	Class	Instance		
01hex	Required	Required	Get_Attributes_All	Gets all attributes of the specified instance.
02hex	N/A	Required	Set_Attributes_All	Sets all attributes of the specified instance.
08hex	Required	N/A	Create	Creates a new connection instance.
09hex	Required	Required	Delete	Deletes an existing connection instance.
0Ehex	Optional	Required	Get_Attribute_Single	Returns the contents of the specified attribute.
10hex	Required	Optional	Set_Attribute_Single	Modifies an attribute value.
15hex	Required	Required	Restore	Restore current connection attributes.

See Appendix A for description of these services

#### 5-48.4.1 Get\_Attributes\_All (Service Code 0x01)

At the **Class level**, the order of the attributes returned in the “Object/service specific reply data” portion of the Get\_Attributes\_All response is as follows:

**Table 5-48.10 Get\_Attributes\_All Response Data – Class Level**

Name	Data Type	Description
Revision	UINT	Object revision
Max Instance	UDINT	Maximum instance number
Num Instances	UDINT	Number of connections currently instantiated
Format Number	UDINT	The format value that shall be specified in the format field of each instance of attribute 9
Edit signature	UDINT	Value created and used by configuration software to detect modification to the instance attribute values

Connection Configuration Object, Class Code: F3<sub>hex</sub>

At the **Instance level**, the order of the attributes returned in the “Object/service specific reply data” portion of the Get\_Attributes\_All response is as follows:

Table 5-48.11 Get\_Attributes\_All Response Data – Instance Level

Name	Data Type	Description
Connection Status	Struct of	Connection status information. When the connection is not open, this attribute contains an error code indicating the reason.
	USINT	General status.
	USINT	Pad.
	UINT	Extended status.
Connection Flags	WORD	Connection Flags.
Target Device Id	Struct of	Target Device identification.
	UINT	Vendor Id
	UINT	Product Type
	UINT	Product Code
	USINT	Major Revision
	USINT	Minor Revision
CS Data Index Number	UDINT	ControlNet Scheduling object read data connection_index number. The default value used when this attribute is not supported shall be 0xFFFFFFFF.
Net Connection Parameters	Struct of	Network connection parameters for both originator-to-target and target-to-originator directions.
	USINT	The connection time-out multiplier.
	BYTE	Transport class and trigger.
	UDINT	Originator-to-target RPI.
	UINT	Originator-to-target network connection parameters.
	UDINT	Target-to-originator RPI.
	UINT	Target-to-originator network connection parameters.
Connection Path	Struct of	The connection path.
	USINT	Size of connection path, in 16-bit words, used in the Connection Manager Forward Open request service
	USINT	Reserved. Shall be zero
	Array of UINT	The connection path. The open connection path size is the length of this array.
Config #1 Data	Struct of	Config #1 data. This data is sent to devices via the Connection Manager Forward Open request service
	UINT	configuration data length in bytes.
	Array of USINT	configuration data. Padded to an even number of bytes.
Config #2 Data	Struct of	Config #2 data. This data is sent to devices via the Forward_Open service
	UINT	configuration data length in bytes.
	Array of USINT	Module configuration data. Padded to even number of bytes.
Connection Name	Struct of	The connection name.
	USINT	Number of characters in connection name.
	USINT	Pad.
	STRING2	Connection name encoded in UNICODE.

**Connection Configuration Object, Class Code: F3<sub>hex</sub>**

Name	Data Type	Description
I/O Mapping	Struct of	The I/O Mapping.
	UINT	Format number.
	UINT	Attribute size in bytes.
	Array of USINT	Attribute data. Padded to an even number of bytes.
Proxy Device Id	Struct of	Proxy Device identification.
	UINT	Vendor Id
	UINT	Product Type
	UINT	Product Code
	USINT	Major Revision
	USINT	Minor Revision
Safety Parameters	See Safety Volume (Volume 5, Chapter 5)  55 octets total	See CIP Safety Specification (Volume 5, Chapter 5). These 55 octets are included in the Get_Attribute_All response only if either the O=>T or T=>O real time transfer format in the Connection Flags Attribute (attribute 2) indicates the format is Safety.
Connection Disable	BOOL	Bit 0, value of connection disable attribute (attribute 12) if supported. 0, if connection disable attribute is not supported.
Net Connection Parameters Attribute Selection	USINT	Selection of Net Connection Parameters attribute or Large Net Connection Parameters attribute if supported. 0 if Net Connection Parameters Attribute Selection is not supported.
Large Net Connection Parameters	Struct of	Network connection parameters for both O=>T and T=>O.
	USINT	The connection time-out multiplier.
	BYTE	Transport class and trigger.
	UDINT	O->T RPI
	UDINT	O->T network connection parameters
	UDINT	T->O RPI
	UDINT	T->O network connection parameters
Additional Safety Parameters	See Safety Volume (Volume 5, Chapter 5)  Variable length.	See CIP Safety Specification (Volume 5, Chapter 5). This group of parameters may be included in the Get_Attribute_All response only if either the O=>T or T=>O real time transfer format in the Connection Flags Attribute (attribute 2) indicates the format is Safety.

**5-48.4.2 Set\_Attributes\_All (Service Code 0x02)**

This service modifies the contents of the attributes shown in Table 5-48.13 and supports the following error codes in addition to other error codes listed in Appendix B:

**Table 5-48.12 Set\_Attributes\_All Service Request Errors**

General Status	Extended Status	Error Name	Description
0x0C	None	Object State Conflict	The Connection Configuration Object cannot accept the Set_Attribute_Single service when an edit session is not active

The structure of the Set\_Attributes\_All Request is shown in the table below.

**Table 5-48.13 Set\_Attributes\_All Service Request**

Name	Data Type	Description
Connection Flags	WORD	Connection flags.
Target Device Id	Struct of	Target Device identification.
	UINT	Vendor Id
	UINT	Product Type
	UINT	Product Code
	USINT	Major Revision
	USINT	Minor Revision
CS Data Index Number	UDINT	ControlNet Scheduling object read data connection_index number. The default value used when this attribute is not supported shall be 0xFFFFFFFF.
Net Connection Parameters	Struct of	Network connection parameters for both O⇒T and T⇒O directions.
	USINT	The connection time-out multiplier.
	BYTE	Transport class and trigger.
	UDINT	O⇒T RPI.
	UINT	O⇒T network connection parameters.
	UDINT	T⇒O RPI.
	UINT	T⇒O network connection parameters.
Connection Path	Struct of	The connection path.
	USINT	Size of connection path, in 16-bit words, used in the Connection Manager Forward Open request service
	USINT	reserved. shall be zero.
	Array of UINT	The connection path. The open connection path size is the length of this array.
Config #1 Data	Struct of	Config #1 data. This data is sent to devices via the Connection Manager Forward Open request service
	UINT	Module configuration data length in bytes.
	Array of USINT	Module configuration data. Padded to even number of bytes.
Config #2 Data	Struct of	Config #2 data. This data is sent to devices via the Connection Manager Forward Open request service
	UINT	Module configuration data length in bytes.
	Array of USINT	Module configuration data. Padded to even number of bytes.

Connection Configuration Object, Class Code: F3<sub>hex</sub>

Name	Data Type	Description
Connection Name	Struct of	The connection name.
	USINT	Number of characters in connection name.
	USINT	Pad.
	STRING2	Connection name encoded in UNICODE.
I/O Mapping	Struct of	The I/O Mapping.
	UINT	Format number.
	UINT	Attribute size in bytes.
	Array of USINT	Attribute data. Padded to an even number of bytes.
Proxy Device Id	Struct of	Proxy Device identification.
	UINT	Vendor Id
	UINT	Product Type
	UINT	Product Code
	USINT	Major Revision
	USINT	Minor Revision
Safety Parameters	See Safety Volume (Volume 5, Chapter 5) 49 octets total	See CIP Safety Specification (Volume 5, Chapter 5). These 49 octets are included in the Set_Attribute_All service data only if either the O=>T or T=>O real time transfer format in the Connection Flags Attribute (attribute 2) indicates the format is Safety.
Connection Disable	BOOL	Bit 0, value of connection disable attribute (attribute 12).
Net Connection Parameters Attribute Selection	USINT	Selection of Net Connection Parameters attribute or Large Net Connection Parameters attribute if supported.
Large Net Connection Parameters	Struct of	Network connection parameters for both O=>T and T=>O.
	USINT	The connection time-out multiplier.
	BYTE	Transport class and trigger.
	UDINT	O->T RPI
	UDINT	O->T network connection parameters
	UDINT	T->O RPI
	UDINT	T->O network connection parameters
Additional Safety Parameters	See Safety Volume (Volume 5, Chapter 5) Variable length.	See CIP Safety Specification (Volume 5, Chapter 5). This group of parameters may be included in the Set_Attribute_All response only if either the O=>T or T=>O real time transfer format in the Connection Flags Attribute (attribute 2) indicates the format is Safety.

**5-48.4.3 Create (Service Code 0x08)**

This service creates a new instance of a Connection Configuration object. Initial attribute values may also be specified with this service. The created instance number is assigned by the class and returned to the requestor. The following request parameters are defined for this service:

**Table 5-48.14 Create Service Request**

Parameter	Data Type	Description
Connection Flags	WORD	Connection flags
NumConnParams	UINT	Number of attribute/value pairs that follow.
ConnParams	Array of Struct of	List of attribute number/value pairs
	UINT	Attribute number
	Object/class attribute-specific Struct	Attribute value

This service supports the following error codes in addition to other error codes listed in Appendix B:

**Table 5-48.15 Create Service Request Errors**

General Status	Extended Status	Error Name	Description
0x02	0x0001	Resource unavailable	The maximum number of instances already exist
	0x0002	Resource unavailable	Not enough memory on device
0x03	None	Invalid parameter value	The attribute count is invalid
0x08	None	Service not supported	Unimplemented service
0x0C	None	Object State conflict	The Connection Configuration Object cannot execute the Set_Attribute_Single service when an edit session is not active
0x0E	None	Attribute not settable	Attempt to set a read-only attribute
0x13	None	Not enough data	The request was too short or truncated
0x1C	None	Missing attribute list entry data	The required attribute was missing

#### 5-48.4.4 Delete (Service Code 0x09)

This service deletes existing connection instances. If addressed to the class-level, all connection instances are deleted. If addressed to the instance-level, only the addressed instance is deleted. This service supports the following error codes in addition to other codes listed in Appendix B:

**Table 5-48.16 Delete Service Request Errors**

General Status	Extended Status	Error Name	Description
0x0C	None	Object State conflict	The Connection Configuration Object cannot execute the Delete service when an edit session is not active

#### 5-48.4.5 Restore (Service Code 0x15)

This service shall discard modifications to instances that have not yet been committed by the Change\_Complete service. If the Restore service is addressed to the class (instance 0), then pending modifications for all instances shall be discarded and the edit session shall be ended. If addressed to a specific instance, only modifications for that instance shall be discarded and the edit session shall not be ended.

**Connection Configuration Object, Class Code: F3<sub>hex</sub>**

This service supports the following error codes in addition to other error codes listed in Appendix B:

**Table 5-48.17 Restore Service Request Errors**

General Status	Extended Status	Error Name	Description
0x0C	None	Object State conflict	The Connection Configuration Object cannot execute the Restore service when an edit session is not active

**5-48.4.6 Object-specific Services**

The Connection Configuration Object provides the following Object Specific Services:

**Table 5-48.18 Connection Configuration Object Object-specific Services**

Service Code	Need in Implementation		Service Name	Description of Service
	Class	Instance		
4Bhex	Required	N/A	Kick_Timer	Kicks Edit Watchdog Timer
4Chex	Conditional*	Conditional*	Open_Connection	Opens connections
4Dhex	Conditional*	Conditional*	Close_Connection	Closes connections
4Ehex	Conditional*	Conditional*	Stop_Connection	Stops connections
4Fhex	Required	N/A	Change_Start	Manages session editing
50hex	Required	N/A	Get_Status	Get status for multiple connections
51hex	Required	N/A	Change_Complete	Completes session editing
52hex	Required	N/A	Audit_Changes	Audits pending changes

\*The Open\_Connection and Close\_Connection services shall either both be supported or neither the Open\_Connection or Close\_Connection services shall be supported. The Stop\_Connection service may only be supported if the Open\_Connection service is supported

**5-48.4.7 Kick\_Timer (Service Code 0x4B)**

This service shall reinitialize the edit watchdog timer. Upon successful execution of the Change\_Start service, the edit watchdog timer shall be started with a period of 60 seconds. This timer is used to recover from the loss of a configuration client between the Change\_Start and Change\_Complete/Restore operations. Receipt of any service request shall reset the edit watchdog timer. Clients may request this service to reset the timer without otherwise affecting the state of the Connection Configuration object. If the edit watchdog timer expires, all pending modifications shall be discarded and the edit session shall be ended.

**5-48.4.8 Open\_Connection (Service Code 0x4C)**

The Open\_Connection service shall cause the connection associated with an instance of the Connection Configuration object to open. If the Open\_Connection service is addressed to the class (instance 0), then all connection instances shall be opened. If addressed to a specific instance, then only that connection instance shall be opened.

**5-48.4.9 Close\_Connection (Service Code 0x4D)**

The Close\_Connection service shall cause the connection associated with an instance of the Connection Configuration object to close. If the Close\_Connection service is addressed to the class (instance 0), then all connection instances shall be closed. If addressed to the instance level, then only the specified instance shall be closed. The Close\_Connection service shall initiate a “graceful” connection shutdown, that is, a Forward\_Close request shall be sent to the connection target. Once a connection has been closed by this service it shall remain closed until an Open\_Connection service is issued.

**5-48.4.10 Stop\_Connection (Service Code 0x4E)**

The Stop\_Connection service shall cause the connection associated with an instance of the Connection Configuration object to stop producing data immediately without sending a Forward\_Close request to the connection target. If the Stop\_Connection service is addressed to the class (instance 0), then all connection instances shall be stopped. If addressed to the instance level, then the specified instance shall be stopped. Once a connection has been stopped, it remains stopped until a subsequent Open\_Connection service request is issued.

**5-48.4.11 Change\_Start (Service Code 0x4F)**

This service shall:

- signal the beginning of an edit session;
- synchronize the current and pending attributes;
- place all connections in the "changeable" state;
- start the edit watchdog timer.

Change\_Start shall be requested prior to performing any services that modify the attributes of a connection. This service shall only be addressed to the class-level (instance 0). If a Change\_Start service is received while an edit session is active, an Object State Conflict error (0x0C) shall be returned.

This service supports the following error codes **in addition to other error codes listed in Appendix B:**

**Table 5-48.19 Change\_Start Service Request Error Codes**

General Status	Extended Status	Error Name	Description
0x0C	None	Object State conflict	An edit session is already active
0x10	None	Device State conflict	The device is in a state that prevents starting an edit session

**5-48.4.12 Get\_Status (Service Code 0x50)**

The Get\_Status service shall retrieve the status attribute (attribute 1) for multiple connections via a single transaction. This service shall be supported at the class-level (instance 0) only. Given a starting instance number, the Get\_Status service shall return instance/status pairs until either the response buffer is full or the status of all connections has been returned.

**Connection Configuration Object, Class Code: F3<sub>hex</sub>**

The following request parameter is defined:

**Table 5-48.20 Get\_Status Service Parameter**

Parameter	Data Type	Description
Starting Instance	UDINT	Starting instance number

The following response parameters are defined for this service:

**Table 5-48.21 Get\_Status Service Response**

Parameter	Data Type	Description
Done Indicator	UINT	0 = More status to be retrieved 1 = All connection status information has been retrieved. All other values reserved.
NumStatusEntries	UINT	Number of instance/status pairs that follow.
StatusEntries	Array of Struct of	List of instance/status pairs
	UDINT	Connection Configuration instance number
	USINT	General status
	USINT	Reserved, shall be 0
	UINT	Extended status

This service supports the following error codes **in addition to other error codes listed in Appendix B:**

**Table 5-48.22 Get\_Status Service Errors**

General Status	Extended Status	Error Name	Description
0x03	None	Invalid Parameter Value	The attribute count is invalid

**5-48.4.13 Change\_Complete (Service Code 0x51)**

This service shall signal the completion of an edit session. Pending attributes for all modified connection instances shall be applied. This service shall take a parameter indicating the type of change being performed; either full or incremental. If an incremental edit is specified, then only the connections that have been modified shall be broken and re-established. If a full edit is specified, then all connections shall be broken and all supporting resources shall be freed and reallocated before attempting to re-establish the connections. The Change\_Complete service shall be supported at the class-level (instance 0) only.

If optional instance attribute 12 is supported and the value of instance attribute 12 is FALSE or if optional instance attribute 12 is not supported an attempt to open the connection shall be made. If optional instance attribute 12 is supported and the value of instance attribute 12 is TRUE, no attempt shall be made to open the connection.

The following request parameter is defined:

**Table 5-48.23 Change\_Complete Service Parameter**

Parameter	Data Type	Description
Change Type	UINT	0 = Full 1 = Incremental All other values reserved.

This service supports the following error codes **in addition to other error codes listed in Appendix B:**

**Table 5-48.24 Change\_Complete Service Errors**

General Status	Extended Status	Error Name	Description
0x02	None	Resource unavailable	Indicates that there is not enough memory on the host device to support the specified configuration.
0x0C	None	Object State conflict	An edit session is not active
0x10	None	Device state conflict	The device is in a state that prevents completing an edition session

**5-48.4.14 Audit\_Changes (Service Code 0x52)**

This service shall verify whether or not there is enough memory on the host device to support a proposed configuration. Like the Change\_Complete service, this service shall take a parameter indicating the type of change being performed; either full or incremental. The Audit\_Changes service shall be supported at the class-level (instance 0) only. This service allows a configuration client to determine if all pending changes will be successful before actually committing the changes with the Change\_Complete service.

The following request parameter is defined:

**Table 5-48.25 Audit\_Changes Service Parameter**

Parameter	Data Type	Description
Change Type	UINT	0 = Full 1 = Incremental All other values reserved.

This service supports the following error codes in addition to other error codes listed in Appendix B:

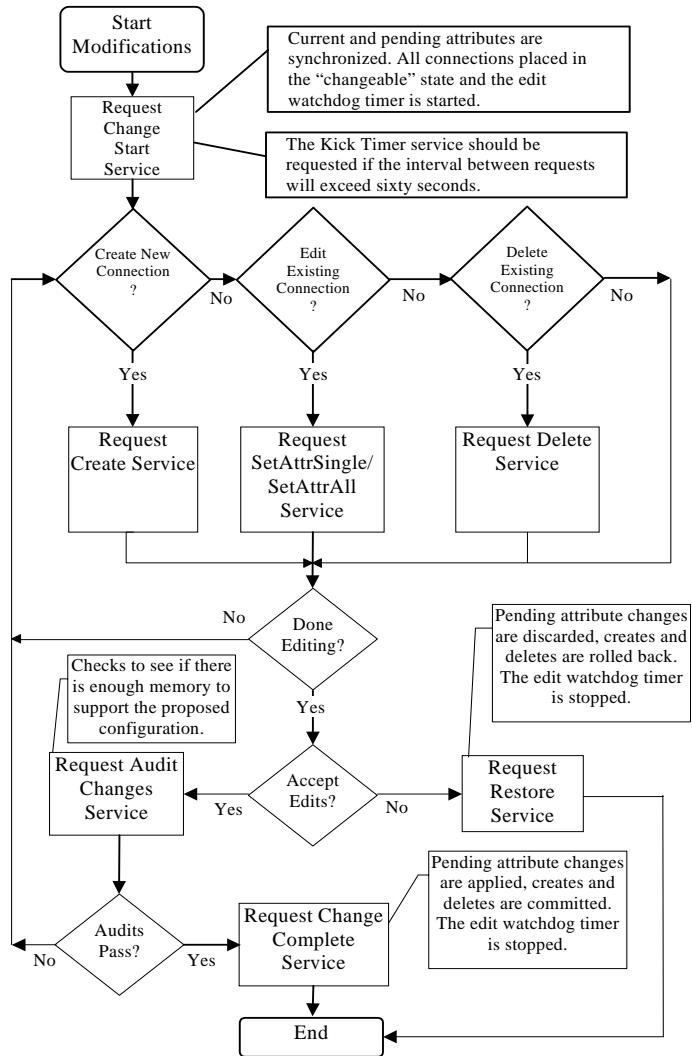
**Table 5-48.26 Audit\_Changes Service Errors**

General Status	Extended Status	Error Name	Description
0x02	None	Resource unavailable	Indicates that there is not enough memory on the host device to support the specified configuration.
0x0C	None	Object State conflict	An edit session is not active
0x10	None	Device state conflict	The device is in a state that prevents completing an audit

## 5-48.5 Behavior

The flowchart below summarizes the process of creating, editing, and deleting connections.

**Figure 5-48.1 Connection Configuration Object Edit Flowchart**



This page is intentionally left blank

# **Volume 1: Common Industrial Protocol (CIP<sup>TM</sup>) Specification**

## **Chapter 6: Device Profiles**

---

## Contents

6-1	Introduction.....	11
6-2	The Object Model.....	11
6-2.1	All Objects Present in a Device .....	12
6-2.2	Objects That Affect Behavior .....	14
6-2.3	Object Interfaces .....	14
6-3	I/O Data Format .....	14
6-3.1	I/O Assembly Instances .....	15
6-3.2	Format of I/O Assembly Data Attribute.....	15
6-3.3	Map of I/O Assembly Data Attribute Components.....	15
6-4	Device Configuration.....	16
6-4.1	Parameter Data.....	17
6-4.2	Effect of Configuration Parameters on Behavior.....	17
6-4.3	Parameter Groups.....	17
6-4.4	Public Interfaces to Device Configuration Data .....	18
6-4.4.1	Parameter Object.....	18
6-4.4.2	Configuration Assembly Object.....	18
6-5	Extended Device Profiles.....	19
6-6	Device Profile Numbering Scheme.....	20
6-7	Device Profiles.....	21
6-8	Generic Device .....	23
6-8.1	Object Model .....	23
6-8.2	How Objects Affect Behavior.....	24
6-8.3	Defining Object Interfaces .....	24
6-9	Limit Switch Device .....	25
6-9.1	Object Model .....	25
6-9.2	How Objects Affect Behavior.....	26
6-9.3	Defining Object Interfaces .....	26
6-9.4	I/O Assembly Instances .....	26
6-9.5	I/O Assembly Data Attribute Format.....	26
6-9.6	Mapping I/O Assembly Data Attribute Components.....	27
6-9.7	Defining Device Configuration.....	27
6-9.7.1	Parameter Object Instances .....	27
6-9.7.2	Mapping Parameter Object Data.....	27
6-9.7.3	Configuration Parameter Definitions .....	28
6-9.8	Effect of Configuration Parameters on Behavior.....	28
6-10	Inductive Proximity Switch Device .....	29
6-10.1	Object Model .....	29
6-10.2	How Objects Affect Behavior.....	30
6-10.3	Defining Object Interfaces .....	30
6-10.4	I/O Assembly Instances .....	30
6-10.5	I/O Assembly Data Attribute Format.....	30
6-10.6	Mapping I/O Assembly Data Attribute Components.....	31
6-10.7	Defining Device Configuration.....	31
6-10.7.1	Parameter Object Instances .....	31
6-10.7.2	Mapping Parameter Object Data.....	31
6-10.7.3	Configuration Parameter Definitions .....	32
6-10.8	Effect of Configuration Parameters on Behavior.....	32
6-11	Photoelectric Sensor Device .....	33
6-11.1	Object Model .....	33
6-11.2	How Objects Affect Behavior.....	34
6-11.3	Defining Object Interfaces .....	34

6-11.4	I/O Assembly Instances .....	34
6-11.5	I/O Assembly Data Attribute Format .....	34
6-11.6	Mapping I/O Assembly Data Attribute Components .....	35
6-11.7	Defining Device Configuration.....	35
6-11.7.1	Parameter Object Instances .....	35
6-11.7.2	Mapping Parameter Object Data.....	35
6-11.7.3	Configuration Parameter Definitions .....	36
6-11.8	Effect of Configuration Parameters on Behavior.....	36
6-12	General Purpose Discrete I/O Device .....	37
6-12.1	Object Model .....	37
6-12.2	How Objects Affect Behavior.....	39
6-12.3	Defining Object Interfaces .....	39
6-12.4	I/O Assembly Instances .....	39
6-12.5	I/O Assembly Data Attribute Format.....	41
6-12.6	Mapping I/O Assembly Data Attribute Components .....	49
6-12.7	Defining Device Configuration.....	50
6-12.7.1	Input Configuration.....	50
6-12.8	Output Configuration Assembly Instances .....	50
6-12.9	Output Configuration Assembly Data Attribute Format.....	51
6-12.10	Mapping Output Configuration Assembly Data Attribute Components.....	51
6-13	Communications Adapter Device .....	53
6-13.1	Object Model .....	53
6-14	General Purpose Analog I/O Device .....	55
6-15	AC/DC Drive Devices .....	56
6-15.1	Multiple axes on one drive.....	56
6-15.2	Object Model .....	57
6-15.3	How Objects Affect Behavior.....	59
6-15.4	Defining Object Interfaces .....	59
6-15.5	I/O Assembly Instances .....	60
6-15.5.1	Connection Paths to I/O Assembly Instances .....	61
6-15.6	I/O Assembly Data Attribute Format .....	61
6-15.7	Mapping I/O Assembly Data Attribute Components .....	63
6-15.8	Defining Device Configuration.....	64
6-15.8.1	Mapping Parameter Object Data.....	65
6-15.8.2	Parameter Group Objects .....	66
6-16	Servo Drive Device .....	69
6-17	Barcode Scanner Device .....	70
6-18	Position Controller Device .....	71
6-18.1	Object Model .....	71
6-18.1.1	Model Description .....	71
6-18.2	How Objects Affect Behavior.....	72
6-18.3	Defining Object Interfaces .....	73
6-18.4	I/O Connection Messages .....	73
6-18.4.1	Message Formats.....	73
6-18.4.2	Definition of a Profile Move .....	74
6-18.4.3	Starting a Profile Move .....	74
6-18.4.4	I/O Handshaking Procedure .....	75
6-18.5	I/O Connection Message Types .....	79
6-18.5.1	Command Message Types .....	79
6-18.5.2	Semantics .....	82
6-18.5.2.1	Load Data/ Start Profile .....	82
6-18.5.2.2	Start Block .....	82
6-18.5.2.3	Incremental .....	82
6-18.5.2.4	Direction (V. Mode) .....	82

6-18.5.2.5	Smooth Stop.....	82
6-18.5.2.6	Hard Stop .....	82
6-18.5.2.7	Reg Arm.....	82
6-18.5.2.8	Enable .....	82
6-18.5.2.9	Block #.....	82
6-18.5.2.10	Command Message Type.....	82
6-18.5.2.11	Response Message Type.....	83
6-18.5.2.12	Command Axis Number .....	83
6-18.5.2.13	Target Position - Command Message Type 01 hex .....	83
6-18.5.2.14	Target Velocity - Command Message Type 02 hex.....	83
6-18.5.2.15	Acceleration - Command Message Type 03 hex .....	83
6-18.5.2.16	Deceleration - Command Message Type 04 hex .....	83
6-18.5.2.17	Torque - Command Message Type 05 hex .....	83
6-18.5.2.18	Attribute Value – Command Message Types 19 – 1E hex .....	83
6-18.5.2.19	Object Attribute to Get – Command Message Types 19 – 1D hex.....	83
6-18.5.2.20	Object Attribute to Set – Command Message Types 19 – 1D hex .....	83
6-18.5.2.21	Command Block Attribute to Get/Set – Command Message Type 1E hex .....	84
6-18.5.2.22	Command Block Instance to Get/Set – Command Message Type 1E hex .....	84
6-18.5.2.23	Parameter Instance to Get - Command Message Type 1F hex .....	84
6-18.5.2.24	Parameter Instance to Set - Command Message Type 1F hex .....	84
6-18.5.2.25	Parameter Value - Command Message Type 1F hex.....	84
6-18.5.3	Response Message Types.....	84
6-18.5.4	Semantics .....	88
6-18.5.4.1	Profile in Progress.....	88
6-18.5.4.2	Block in Execution .....	88
6-18.5.4.3	On Target Position .....	88
6-18.5.4.4	General Fault .....	88
6-18.5.4.5	Current Direction .....	88
6-18.5.4.6	Home Level .....	88
6-18.5.4.7	Reg Level.....	88
6-18.5.4.8	Enable .....	88
6-18.5.4.9	Executing Block #.....	89
6-18.5.4.10	Fault Input Fault .....	89
6-18.5.4.11	Fwd Limit .....	89
6-18.5.4.12	Rev Limit .....	89
6-18.5.4.13	Positive Limit .....	89
6-18.5.4.14	Negative Limit .....	89
6-18.5.4.15	FE Fault .....	89
6-18.5.4.16	Block Fault.....	89
6-18.5.4.17	Load Complete.....	89
6-18.5.4.18	Response Message Type .....	89
6-18.5.4.19	Response Axis Number .....	89
6-18.5.4.20	Actual Position - Response Message Type 01 hex .....	90
6-18.5.4.21	Commanded Position - Response Message Type 02 hex .....	90
6-18.5.4.22	Actual Velocity - Response Message Type 03 hex.....	90
6-18.5.4.23	Command Velocity - Response Message Type 04 hex.....	90
6-18.5.4.24	Torque - Response Message Type 05 hex .....	90
6-18.5.4.25	Home Position - Response Message Type 06 hex .....	90
6-18.5.4.26	Index Position - Response Message Type 07 hex.....	90
6-18.5.4.27	Registration Position - Response Message Type 08 hex .....	90
6-18.5.4.28	General Error Code – Response Message Type 14 hex.....	91
6-18.5.4.29	Additional Code – Response Message Type 14 hex.....	91
6-18.5.4.30	Attribute Value – Response Message Types 19 – 1E hex .....	91
6-18.5.4.31	Object Attribute to Get – Response Message Types 19 – 1E hex .....	91
6-18.5.4.32	Parameter Instance to Get - Response Message Type 1F hex .....	91

6-18.5.4.33 Parameter Value - Response Message Type 1F hex .....	91
6-18.6 Mapping I/O Message Data Attribute Components .....	92
6-19 Motor Overload Device .....	94
6-19.1 Object Model .....	94
6-19.2 How Objects Affect Behavior .....	95
6-19.3 Defining Object Interfaces .....	96
6-19.4 Contactor Interface and Behavior .....	96
6-19.5 I/O Assembly Instances .....	97
6-19.5.1 Connection Paths to I/O Assembly Instances .....	97
6-19.6 I/O Assembly Data Attribute Format .....	98
6-19.6.1 Output Assembly Data Attribute Format .....	98
6-19.6.2 Input Assembly Data Attribute Format .....	98
6-19.7 Mapping I/O Assembly Data Attribute Components .....	98
6-19.7.1 Mapping for Output Assembly Data Components .....	98
6-19.7.2 Mapping for Input Assembly Data Components .....	99
6-19.8 Defining Device Configuration .....	99
6-20 Weigh Scale Device .....	100
6-21 Encoder Device .....	101
6-21.1 Introduction .....	101
6-21.2 Object Model .....	101
6-21.3 How Objects Affect Behavior .....	102
6-21.4 Defining Object Interfaces .....	102
6-21.5 I/O Assembly Instances .....	102
6-21.6 I/O Assembly Data Attribute Format .....	103
6-21.7 Mapping I/O Assembly Data Attribute Components .....	104
6-21.8 Defining Device Configuration .....	104
6-21.9 Mapping Parameter Object Data .....	104
6-22 Resolver Device .....	106
6-22.1 Object Model .....	106
6-22.2 How Objects Affect Behavior .....	107
6-22.3 Defining Object Interfaces .....	107
6-22.4 I/O Assembly Instances .....	107
6-22.5 I/O Assembly Data Attribute Format .....	108
6-22.6 Mapping I/O Assembly Data Attribute Components .....	108
6-22.7 Configuration Assembly Instances .....	108
6-22.8 Configuration Assembly Data Attribute Format .....	109
6-22.9 Mapping Configuration Assembly Data Attribute Components .....	109
6-22.10 Defining Device Configuration .....	109
6-22.10.1 Parameter Object Instances .....	110
6-22.10.2 Mapping Parameter Object Data .....	110
6-22.10.3 Configuration Parameter Definitions .....	111
6-22.11 Effect of Configuration Parameters on Behavior .....	112
6-23 Control Station Device .....	113
6-24 Message Display Device .....	114
6-25 Circuit Breaker .....	115
6-26 Pneumatic Valve Device .....	116
6-26.1 Object Model .....	116
6-26.2 How Objects Affect Behavior .....	117
6-26.3 Defining Object Interfaces .....	117
6-26.4 I/O Assembly Instances .....	117
6-26.5 I/O Assembly Data Attribute Format .....	119
6-26.5.1 The I/O Assembly data attribute for <i>Solenoids Status Data</i> is shown below: .....	119
6-26.5.2 The I/O Assembly data attribute for <i>General Purpose Discrete Input Data</i> is shown below: .....	121
6-26.5.3 The I/O Assembly data attribute for <i>Solenoids Status Data</i> plus <i>Discrete Input Data</i> .....	122

6-26.5.4	The I/O Assembly data attribute for the <i>Solenoid Valves Output Data</i> .....	125
6-26.6	Mapping I/O Assembly Data Attribute Components .....	126
6-26.7	Configuration Data and Public Interfaces .....	127
6-26.8	Parameter Object Instances .....	127
6-27	Contactor Device .....	128
6-27.1	Object Model .....	128
6-27.2	How Objects Affect Behavior .....	129
6-27.3	Defining Object Interfaces .....	129
6-27.4	Contactor Interface and Behavior .....	130
6-27.5	I/O Assembly Instance .....	131
6-27.5.1	Connection Paths to I/O Assembly Instances .....	131
6-27.6	I/O Assembly Data Attribute Format .....	132
6-27.7	Mapping I/O Assembly Data Attribute Components .....	132
6-27.8	Defining Device Configuration .....	132
6-28	Motor Starter Device .....	133
6-28.1	Object Model .....	133
6-28.2	How Objects Affect Behavior .....	134
6-28.3	Defining Object Interfaces .....	134
6-28.3.1	Starter Interface and Behavior .....	135
6-28.3.2	Reversing Motor Starter Interface and Behavior .....	136
6-28.3.3	Two Speed Motor Starter Interface and Behavior .....	137
6-28.4	I/O Assembly Instances .....	139
6-28.4.1	Connection Paths to I/O Assembly Instances .....	139
6-28.5	I/O Assembly Data Attribute Format .....	140
6-28.5.1	Output Assembly Data Attribute Format .....	140
6-28.5.2	Input Assembly Data Attribute Format .....	140
6-28.6	Mapping I/O Assembly Data Attribute Components .....	141
6-28.6.1	Mapping for Motor Starter Output Assembly Data Components .....	141
6-28.6.2	Mapping for Motor Starter Input Assembly Data Components .....	141
6-28.7	Defining Device Configuration .....	141
6-29	Softstart Starter Device .....	142
6-29.1	Object Model .....	142
6-29.2	How Objects Affect Behavior .....	143
6-29.3	Defining Object Interfaces .....	144
6-29.4	Softstart Motor Interface and Behavior .....	145
6-29.5	I/O Assembly Instances .....	147
6-29.5.1	Connection Paths to I/O Assembly Instances .....	147
6-29.6	I/O Assembly Data Attribute Format .....	148
6-29.6.1	Output Assembly Data Attribute Format .....	148
6-29.6.2	Input Assembly Data Attribute Format .....	148
6-29.7	Mapping I/O Assembly Data Attribute Components .....	148
6-29.8	Defining Device Configuration .....	148
6-30	Human-Machine Interface (HMI) .....	149
6-30.1	Object Model .....	149
6-30.2	How Objects Affect Behavior .....	150
6-30.3	Defining Object Interfaces .....	150
6-31	Mass Flow Controller Device .....	151
6-31.1	Object Model .....	151
6-31.1.1	Class Subclasses .....	151
6-31.1.2	Instance Subclasses .....	151
6-31.2	How Objects Affect Behavior .....	152
6-31.3	Defining Object Interfaces .....	153
6-31.4	I/O Assembly Instances .....	154
6-31.5	I/O Assembly Object Instance Data Attribute Format .....	155

6-31.6	Mapping I/O Assembly Data Attribute Components .....	158
6-31.7	Object Limitations and Specific Definitions .....	159
6-31.7.1	S-Device Supervisor Object Instance .....	159
6-31.7.1.1	Limitations .....	159
6-31.7.1.2	Specific Definition .....	159
6-31.7.2	S-Analog Sensor Object Instance .....	160
6-31.7.2.1	Limitations .....	160
6-31.7.3	S-Analog Actuator Object Instance .....	160
6-31.7.3.1	Limitations .....	160
6-31.7.4	S-Single Stage Controller Object Instance .....	161
6-31.7.4.1	Limitations .....	161
6-31.8	Defining Device Configuration.....	161
6-32	Vacuum/Pressure Gauge Device.....	163
6-32.1	Object Model .....	163
6-32.2	Class Subclasses.....	164
6-32.3	Instance Subclasses .....	164
6-32.4	How Objects Affect Behavior.....	166
6-32.5	Defining Object Interfaces.....	167
6-32.6	I/O Assembly Instances .....	167
6-32.6.1	Mapping I/O Assembly Data Attribute Components .....	173
6-32.7	Object Limitations and Specific Definitions .....	174
6-32.7.1	S-Device Supervisor Object Instance .....	174
6-32.7.1.1	Exception Attributes Definition.....	174
6-32.7.1.2	Manufacturer's Device Type Attribute Definition.....	176
6-32.7.1.3	Behavior.....	176
6-32.7.2	S-Analog Sensor Object Instance .....	176
6-32.7.2.1	Limitations .....	176
6-32.7.2.1.1	Object Instance Number Assignment .....	176
6-32.7.2.1.2	Data Type, Data Units and Produce Trigger Delta Type .....	177
6-32.7.2.1.3	Alarm and Warning Hysteresis (attributes 19 and 23).....	177
6-32.7.2.2	Object Extensions .....	177
6-32.7.3	Trip Point Object Instance .....	178
6-32.7.3.1	Limitations .....	178
6-32.7.3.1.1	Hysteresis (attribute 10).....	178
6-32.7.3.1.2	Source (attribute 14) .....	178
6-32.7.3.1.3	Destination (attribute 12) .....	178
6-32.8	Defining Device Configuration.....	179
6-33	Programmable Logic Controller Device .....	180
6-33.1	Object Model .....	180
6-33.2	Defining Object Interfaces .....	180
6-34	ControlNet Physical Layer Component Device .....	181
6-35	Process Control Valve Device .....	182
6-35.1	Object Model .....	182
6-35.2	How Objects Affect Behavior.....	184
6-35.3	Defining Object Interfaces .....	185
6-35.3.1	I/O Assembly Instances .....	185
6-35.3.2	Mapping I/O Assembly Data Attribute Components .....	188
6-35.4	Object Limitations and Specific Definitions .....	188
6-35.4.1	S-Device Supervisor Object.....	188
6-35.4.1.1	Exception Attributes Definition.....	189
6-35.4.2	S-Analog Sensor Object.....	190
6-35.4.3	S-Single Stage Controller Object .....	191
6-35.4.4	Selection Object .....	192
6-35.4.5	Discrete Input Point Object.....	193
6-35.5	Defining Device Configuration.....	193

6-36	Turbomolecular Vacuum Pump Device.....	194
6-36.1	Object Model .....	194
6-36.2	Class Subclasses.....	194
6-36.3	Instance Subclasses .....	194
6-36.4	How Objects Affect Behavior.....	196
6-36.5	Defining Object Interfaces.....	196
6-36.6	I/O Assembly Instances .....	197
6-36.7	I/O Assembly Instance Component Mapping.....	199
6-36.8	Object Limitations and Specific Definitions.....	200
6-36.8.1	S-Device Supervisor .....	200
6-36.8.2	Register Object.....	201
6-36.8.2.1	Object Instance Number Assignment .....	201
6-36.8.3	Discrete Input Point Object.....	202
6-36.8.3.1	Object Instance Number Assignment .....	202
6-36.8.3.2	Attribute Definitions .....	202
6-36.8.4	Discrete Output Point Object .....	202
6-36.8.4.1	Object Instance Number Assignment .....	202
6-36.8.5	AC/DC Drive Object.....	203
6-36.8.5.1	Attribute Limitations.....	203
6-36.8.5.2	Behavior.....	204
6-36.8.6	S-Analog Sensor Object.....	204
6-36.8.6.1	Object Instance Number Assignment .....	204
6-36.8.6.2	Attribute Definitions .....	204
6-36.8.7	S-Single Stage Controller Object .....	205
6-36.8.7.1	Object Instance Number Assignment .....	205
6-36.8.7.2	Attribute Definitions .....	205
6-36.9	Defining Device Configuration.....	205
6-37	Residual Gas Analyzer Device .....	206
6-37.1	Object Model .....	206
6-37.2	Class Subclasses.....	206
6-37.3	Instance Subclasses .....	206
6-37.4	How Objects Affect Behavior.....	208
6-37.5	Defining Object Interfaces.....	208
6-37.6	I/O Assembly Instances .....	208
6-37.6.1	I/O Assembly Instance Component Formats .....	210
6-37.6.2	I/O Assembly Instance Component Mapping .....	211
6-37.7	Object Limitations and Specific Definitions.....	211
6-37.7.1	S-Analog Sensor Object.....	212
6-37.7.1.1	Class.....	212
6-37.7.1.2	Instance .....	212
6-37.7.1.3	Attribute Definitions .....	212
6-37.7.2	Discrete Input Point Object Instance .....	213
6-37.7.3	Discrete Output Point Object Instance .....	213
6-37.7.3.1	Object Instance Number Assignment .....	213
6-37.8	Defining Device Configuration.....	213
6-38	DC Power Generator Device .....	214
6-38.1	Object Model .....	214
6-38.2	Subclasses .....	214
6-38.3	How Objects Affect Behavior.....	217
6-38.4	Defining Object Interfaces.....	217
6-38.5	I/O Assembly Instances .....	218
6-38.5.1	I/O Assembly Instance Component Formats .....	220
6-38.5.2	I/O Assembly Instance Component Mapping .....	221
6-38.6	Object Limitations and Specific Definitions.....	222
6-38.6.1	S-Device Supervisor .....	222

6-38.6.2	S-Analog Sensor Object Instances .....	223
6-38.6.3	S-Analog Actuator Object Instances .....	224
6-38.6.4	S-Single Stage Controller Object Instances .....	224
6-38.6.4.1	Behavior.....	224
6-38.6.4.2	Selection Object, Instance 1.....	225
6-38.6.5	Register Object.....	225
6-38.6.5.1	Register Object, Instance 1 (Input) Operational Status.....	225
6-38.6.5.2	Register Object, Instance 2 (Input) Interlock Status .....	226
6-38.6.5.3	Behavior.....	226
6-38.7	Defining Device Configuration.....	226
6-39	RF Power Generator Device .....	227
6-39.1	Object Model .....	227
6-39.2	Subclasses .....	227
6-39.3	How Objects Affect Behavior.....	230
6-39.4	Defining Object Interfaces.....	230
6-39.5	I/O Assembly Instances .....	231
6-39.5.1	I/O Assembly Instance Component Formats .....	233
6-39.5.2	I/O Assembly Instance Component Mapping .....	233
6-39.6	Object Limitations and Specific Definitions.....	235
6-39.6.1	S-Device Supervisor .....	235
6-39.6.2	S-Analog Sensor Object Instances .....	236
6-39.6.3	S-Analog Actuator Object Instances.....	237
6-39.6.3.1	S-Single Stage Controller Object Instances .....	237
6-39.6.3.2	Behavior.....	237
6-39.6.4	Selection Object, Instance 1 .....	238
6-39.6.5	Register Object.....	238
6-39.6.5.1	Register Object, Instance 1 Operational Status.....	238
6-39.6.5.2	Register Object, Instance 2 Interlock Status .....	239
6-39.6.5.3	Behavior.....	239
6-39.7	Defining Device Configuration.....	239
6-40	Fluid Flow Controller Device .....	240
6-40.1	Object Model .....	240
6-40.2	Class Subclasses.....	241
6-40.3	Instance Subclasses .....	241
6-40.4	Instance Identifiers.....	241
6-40.5	How Objects Affect Behavior.....	243
6-40.6	Defining Object Interfaces .....	243
6-40.7	I/O Assembly Instances .....	243
6-40.8	I/O Assembly Object Instance Data Attribute Encoding .....	244
6-40.9	Mapping I/O Assembly Data Attribute Components .....	245
6-40.10	Object Limitations and Specific Definitions .....	245
6-40.10.1	S-Device Supervisor Object Instance .....	246
6-40.10.2	Specific Definition .....	246
6-40.10.3	S-Analog Sensor Object.....	247
6-40.10.4	S-Analog Actuator Object.....	247
6-40.10.5	S-Single Stage Controller Object .....	248
6-40.11	Defining Device Configuration .....	248
6-41	CIP Motion Drive .....	249
6-41.1	Introduction.....	249
6-41.2	Object Model .....	249
6-41.2.1	Object Description .....	249
6-41.3	How Objects Affect Behavior .....	250
6-41.4	Defining Object Interfaces .....	251
6-41.5	I/O Connection Messages .....	251
6-41.5.1	Motion Drive Connection .....	251

6-41.5.1.1	Motion I/O Connection Overview .....	253
6-41.5.1.2	Motion Connection Structure.....	253
6-41.5.2	Controller-to-Device Connection.....	254
6-41.5.2.1	Controller-to-Device Connection Header .....	255
6-41.5.2.2	Instance Data Blocks .....	258
6-41.5.2.3	Event Data Block .....	264
6-41.5.2.4	Service Data Block .....	266
6-41.5.3	Device-to-Controller Connection.....	267
6-41.5.3.1	Device-to-Controller Connection Header .....	267
6-41.5.3.2	Instance Data Blocks .....	270
6-41.5.3.3	Event Data Block .....	273
6-41.5.3.4	Service Data Block .....	275
6-41.5.4	Fixed Drive Connection Format.....	276
6-41.5.5	Motion Connection Timing.....	276
6-41.5.5.1	Controller-to-Device Connection Timing.....	278
6-41.5.5.2	Device-to-Controller Connection Timing.....	282
6-41.5.5.3	Device Update Period Independence .....	284
6-41.5.5.4	Transmission Latency Independence .....	285
6-41.5.5.5	System Time Offset Compensation .....	286
6-41.6	Drive Startup Procedure.....	289
6-41.6.1	Motion I/O Connection Creation .....	289
6-41.6.2	Motion Axis Object Configuration .....	289
6-41.6.3	Time Synchronization .....	289
6-41.7	Device Visualization.....	291
6-41.8	Ethernet Quality of Service (QoS) .....	291
6-42	Enhanced Mass Flow Controller Device .....	292
6-42.1	Introduction.....	292
6-42.2	Object Model .....	292
6-42.3	Class Subclasses.....	293
6-42.4	Instance Subclasses .....	293
6-42.5	How Objects Affect Behavior.....	294
6-42.6	Defining Object Interfaces .....	295
6-42.7	I/O Assembly Instances .....	295
6-42.8	I/O Assembly Object Instance Data Attribute Format .....	296
6-42.8.1	Mapping I/O Assembly Data Attribute Components .....	300
6-42.9	Object Limitations and Specific Definitions .....	301
6-42.9.1	S-Device Supervisor Object Instance.....	301
6-42.9.2	S-Analog Sensor Object.....	302
6-42.9.3	S-Analog Sensor Object.....	303
6-42.9.4	S-Analog Actuator Object.....	303
6-42.9.5	S-Single Stage Controller Object .....	303
6-42.10	Defining Device Configuration.....	304

## **6-1 Introduction**

To provide interoperability and promote interchangeability by like device types, there must be some consistency between devices of the same type. That is, there must be a core “standard” for each device type. In general, like devices must:

- exhibit the same behavior
- produce and/or consume the same basic set of I/O data
- contain the same basic set of configurable attributes

The formal definition of this information is known as a *device profile*. This chapter provides a detailed definition of a device profile and describes its components.

A device profile **shall** contain:

- an object model for the device type
- the I/O data format for the device type configuration data and the public interface(s) to that data

You may adopt or extend one of the existing profiles in this chapter or you may define your own profile based on the format contained in this chapter.

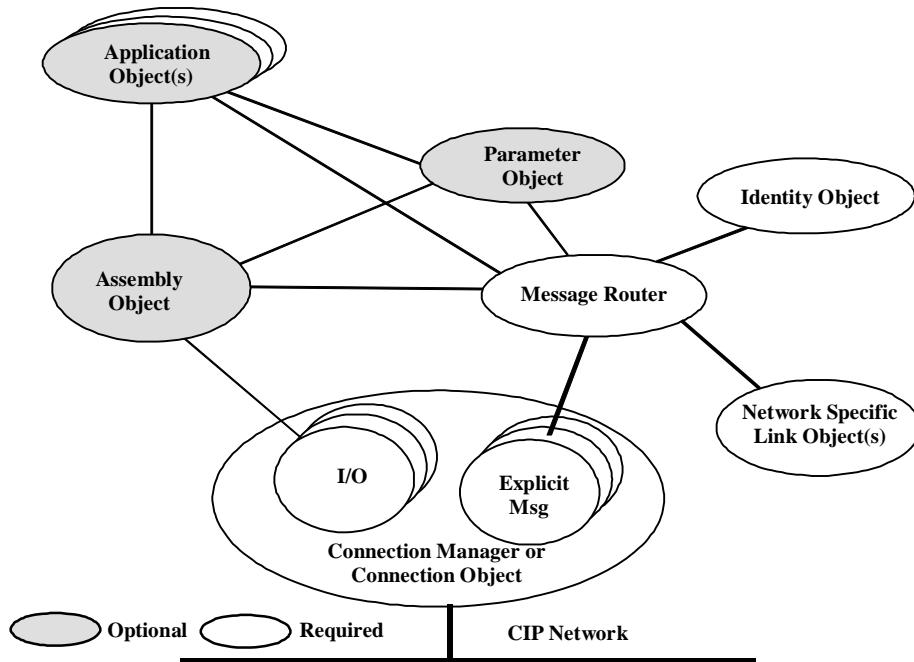
## **6-2 The Object Model**

To provide interoperability among like devices, the same object implemented in two or more devices **shall** behave identically from device to device. Consequently, each object specification includes a rigid definition of behavior.

Every CIP product contains several objects. These objects interact to provide basic product behavior. Because the behavior of individual objects is fixed, the behavior of identical groupings of objects is also fixed. Therefore, the same group of objects arranged in a specified order will interact to produce the same behavior from device to device.

The *grouping* of objects used in a device is referred to as that device’s *object model*. See Figure 6-2.1. For like devices to produce identical behavior, they must have identical object models. Therefore, an object model is included with every device profile to provide for interoperability among like CIP devices.

Figure 6-2.1 Object Model



An object model specification:

- Identifies all object classes present in the device (required, optional and conditional).
- Indicates the number of instances present in each object class. If the device supports the dynamic creation and deletion of instances, then the object model states the maximum number of instances that can exist within the object class.
- States whether or not the object affects behavior of the device. If it does affect behavior, the object model states how.
- Defines the interface to each object. This defines how objects and object classes are linked.

## 6-2.1 All Objects Present in a Device

Every device can contain both required objects, optional, and conditional objects. When an object is identified as “required,” it is required for *all* devices of that type. Each device profile shall contain an Object Interface Table, which shall list the interface to each object. At a minimum, every object model for a CIP common device must specify instances of these object classes:

**Table 6-2.2 Minimum Objects Required For All Devices**

Object Class		Optional/Required	# of Instances
	Identity Object	Required	1
	Message Router Object	Required	1
Network Specific Link Object(s)	DeviceNet Object	Conditional <sup>1</sup>	1
	Ethernet Link Object	Conditional <sup>2</sup>	1
	TCP/IP Interface Object	Conditional <sup>2</sup>	1
	ControlNet Object	Conditional <sup>3</sup>	1
	CompoNet Link Object	Conditional <sup>4</sup>	1
	Connection Object	Conditional <sup>1, 5</sup>	2 (explicit,-I/O) <sup>5</sup>
	Connection Manager Object	Conditional <sup>2 - 3</sup>	1

<sup>1</sup> Required if CIP network type is DeviceNet<sup>2</sup> Required if CIP network type is EtherNet/IP<sup>3</sup> Required if CIP network type is ControlNet<sup>4</sup> Required if CIP network type is CompoNet<sup>5</sup> If CIP network type is CompoNet, see Chapter 6 of Volume 6, CompoNet Adaptation of CIP

Although not an object, each device shall also support the Unconnected Message Manager (UCMM). In addition to these minimum object classes, object models can, and probably will, contain application-specific object classes that are required by the device type.

Some object classes may be included that provide functions beyond the minimum required of a particular device type or that have no effect on device behavior. These types of objects are identified in the profile as “optional” or “conditional.” When an object is identified as “optional,” it is optional for *all* devices of that type. The device type dictates what objects are necessary to provide the device’s required basic function. When an object is identified as “conditional,” it is required “if” a specified condition exists. The Device Profile shall specify the conditions.

**Important:** Instances of OPTIONAL object classes may provide behavior beyond the behavior defined for the device type. At power up, however, this additional behavior shall default in a manner such that the device’s behavior appears to be identical to the basic behavior defined for that device type.

**Table 6-2.3 Required/Optional Object Criteria**

Object classes are specified as REQUIRED when they	Object classes are specified as OPTIONAL when they
affect in any way the basic behavior specified for the device type	provide behavior beyond the minimum specified for the device type
are used to define the I/O data format of the device	provide functions beyond the minimum required of a particular device type or have NO effect on behavior of the device
provide the primary method of access to the device’s configuration data	provide an optional method of access to the device’s configuration data.

## 6-2.2 Objects That Affect Behavior

After all objects included in the device are identified, this section of a profile distinguishes between objects that do and do not affect the behavior of the device. If an object affects behavior, this section states how. Any component (object, attribute, or service) that affects the behavior of a device is specified here. The following table shows the format of this part of the device profile description.

**Table 6-2.4 Components That Affect Behavior of the Device**

Component	Effect on behavior
Attribute/Object	Behavior

## 6-2.3 Object Interfaces

The final portion of the object model specification within a device profile is the definition of all interfaces to each of the device's internal objects. Defining object interfaces indicates how the objects within a device are connected.

**Table 6-2.5 Object Interfaces**

Object	Interface
Name of Object	Name of Interface (Explicit Messaging Connection Instance, Message Router, I/O connection)

In summary, an object model defines behavior of a device in the following terms:

- objects present in device
- maximum number of object instances
- how objects affect behavior
- object interfaces

## 6-3 I/O Data Format

This section of a profile defines how a device communicates on the CIP network, which includes an exact specification of the device's I/O data format.

Smart networked devices can (and probably will) produce and/or consume more than one I/O value. Typically, they will produce and/or consume one or more I/O values, as well as status and diagnostic information. Each piece of data communicated by a device is represented by an attribute of one of the device's internal objects.

Communicating multiple pieces of data (attributes) across a single I/O connection requires that the attributes be grouped or assembled together into a single data block. Instances of the **Assembly Object Class** perform this grouping. Thus, the definition of a device's I/O data format is equivalent to the definition of the assembly instances used to group the device's I/O data.

In a device profile, the I/O data format of devices adheres to these guidelines:

- I/O Assemblies are either **Input type** or **Output type**
- A device may contain more than one I/O assembly (data format of I/O instances may be a configurable option of your device)

The definition of a device's I/O assembly instances:

- Identifies the I/O assembly by instance number, type, and name
- Specifies the I/O assembly Data attribute format
- Maps the I/O assembly Data attribute components to other attributes

### **6-3.1 I/O Assembly Instances**

Because CIP products can contain one or more I/O assemblies (of either Input type or Output type), assembly instances are clearly identified for each device type. The following table identifies the I/O assembly instance supported by a typical device.

**Table 6-3.1 Identifying I/O Assembly Instances**

Number	Type	Name
Assembly #	Input or Output	Name of assembly

### **6-3.2 Format of I/O Assembly Data Attribute**

Any device communicating I/O data to and from another device must have knowledge of the other device's I/O data format. The *Data* attribute of the Assembly Object (instance attribute #3) holds this I/O format. Therefore, this section of a profile specifies the format of the Data attribute for **each** assembly instance listed in the assembly instance identification table.

The Data attribute is an array of bytes. The device profile specifies how that array is defined to represent a device's I/O data.

Specification of the I/O assembly Data attribute format adheres to these guidelines:

- List Data components that are larger than one byte in size with the low-order byte first
- Right justify within a byte (starting with bit 0) Data components that are smaller than one byte
- Explicitly state if bits or bytes are to be reserved

### **6-3.3 Map of I/O Assembly Data Attribute Components**

Because components of the Assembly Object's *Data* attribute are attributes of other objects, a device profile contains a mapping of those attributes to their respective objects.

The map includes specification of the member path (Class ID, Instance ID, etc.) for each data component. Specification of the relative addresses of each Data attribute component is essentially equivalent to specification of the **Member\_List** instance attribute (#2) of the Assembly Object.

The following table shows the format for an I/O assembly Data attribute mapping.

**Table 6-3.2 Example I/O Assembly Data Attribute Mapping**

Data Component Name	Class		Instance Number	Attribute		Data Type
	Name	Number		Name	Number	
Component name within profile	Component class name within object library	xxhex	Y	Component attribute name within object library	z	

If a device has more than one I/O assembly instance, the profile should include a table similar to the one above for **each** I/O assembly instance.

## 6-4 Device Configuration

In addition to a product's object model and format of its I/O data, a device profile includes specification of the device's configurable parameters and the public interface to those parameters.

The configurable parameters in a device directly affect its behavior. Because like devices must behave in an identical fashion, they **must** have identical configuration parameters.

**Important:** “Identical configuration” refers to *basic* configuration. A device may have extended functionality (with associated parameters) that is beyond the behavior defined for the device type. At power up, this functionality must default in a manner such that the device’s behavior appears to be identical to the behavior defined for that device type.

In addition to defining identical configuration parameters, the public interfaces to those parameters **must** be identical.

Definition of a device’s configuration includes the following information for **each** configurable attribute:

- configuration parameter data:
  - all attribute values of each *Parameter Object Instance*
  - all values in the parameter section of an *Electronic Data Sheet*
  - at minimum, the following printed data sheet information:
    - parameter name
    - attribute path (class, instance, attribute)
    - data type
    - parameter units
    - minimum/maximum default values

- effect of parameters on device behavior
- parameter groups if any configurable parameters are grouped using an instance of the Parameter Group Object Class
- public interface to the device's configuration (i.e., bulk configuration via a configuration assembly, full/stub instances of the Parameter Object Class, etc.)

#### **6-4.1 Parameter Data**

The definition of each configuration parameter includes specification of one of the following:

- the instance attributes of an instance of the Parameter Object Class (for each of your configuration parameters)
- all data outlined in the parameter section of an EDS
- at minimum, the following printed data sheet information:
  - parameter name
  - attribute path (class, instance, attribute)
  - data type
  - parameter units
  - minimum/maximum default values

#### **6-4.2 Effect of Configuration Parameters on Behavior**

The effect that each of the configuration parameters has on the device's behavior is also documented in the configuration section of a device profile. The following table shall be used within the device profile.

**Table 6-4.1 Example of Table Showing the Effect of Parameters on Behavior**

Parameter	Effect on Behavior
Parameter name	Effect

#### **6-4.3 Parameter Groups**

If any configurable parameters are grouped using an instance of the *Parameter Group Object Class*, then the definition of each group is specified in this section.

The definition of each configuration parameter group includes specification of either:

- the instance attributes of an instance of the Parameter Group Object Class (for each of your configuration parameters); or
- all data outlined in the parameter group section of an EDS

## 6-4.4 Public Interfaces to Device Configuration Data

The final portion of the configuration section of a profile clearly specifies the public interface(s) to a device's configuration data.

### 6-4.4.1 Parameter Object

If a device employs instances of the Parameter Object Class, each instance and the configuration parameter associated with it is specified here. Also included here is a map of the configuration parameter to the object in which it is contained.

**Table 6-4.2 Parameter Instance Listing**

Instance Number	Configuration Parameter Name
X	Parameter name

**Table 6-4.3 Configuration Parameter Mapping**

Configuration Parameter Name	Class		Instance Number	Attribute		Data Type
	Name	Number		Name	Number	
Parameter name	Class	xxhex	y	Attribute	z	

### 6-4.4.2 Configuration Assembly Object

Documentation of a device's configuration assembly provides information similar to that which is specified for the device's I/O assemblies. This section of a profile includes:

- specification of the configuration assembly Data attribute format
- mapping of each configurable attribute using its logical address (Class/Instance/Attribute)

Specification of the configuration assembly Data attribute format adheres to these guidelines:

- List Data components that are larger than one byte in size with the low-order byte first
- Right justify within a byte (starting with bit 0) Data components that are smaller than one byte
- Explicitly state if bits or bytes are to be reserved

The table below shows how the format of the Configuration Assembly Object's Data attribute is specified.

**Table 6-4.4 Configuration Assembly Data Attribute Format**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Configuration Parameter 1							
1	Configuration Parameter 2							
2	Configuration Parameter 3							

In addition to specification of the device's configuration assembly Data attribute format, this section also includes a mapping of the individual configuration assembly Data attribute components to their respective objects.

The map includes specification of the Class, Instance, and Attribute IDs for each data component. Specification of the relative addresses of each Data attribute component is essentially equivalent to specification of the **Member\_List** instance attribute (#2) of the Assembly Object. The table below shows the format for the configuration assembly Data attribute mapping.

**Table 6-4.5 Configuration Assembly Data Attribute Mapping**

Configuration Parameter Name	Class		Instance Number	Attribute		Data Type
	Name	Number		Name	Number	
Configuration Parameter1	Class	xxhex	y	Range	Z	

## 6-5 Extended Device Profiles

You have the option of adopting existing device profiles and then extending them to incorporate any additional behavior your product may exhibit.

Manufacturers of multiple source products may wish to design a product such that it provides the basic behavior defined in the product's device profile and, in addition, provides extended functionality that helps distinguish one product from another.

**Important:** The basic device profile definition must not change when extending an existing device profile. Also, the added functionality must not make the extended profile incompatible with the basic device profile. For these reasons, you must adhere to the following rules when extending an existing device profile:

- All new objects, attributes, and services added to the profile are OPTIONAL. Backwards compatibility *shall* be maintained.
- At power-up, all new behavior must default such that the device's behavior appears identical to the specified default behavior defined for the device type.
- The basic I/O format must not change. Extended I/O formats can be provided for by adding optional I/O assembly instances.
- The basic configuration must not change. Extended configuration parameters can be provided for by adding optional configuration assembly instances or optional instances of the Parameter Object Class.
- Any additional assembly instances must be defined in the vendor-specific address range.

**Important:** Instances of the Assembly Class are divided into address ranges to provide for extensions to device profiles. See the Assembly Object definition in the object library.

## 6-6

## Device Profile Numbering Scheme

The table below reveals the numbering scheme to be used for device profile numbering. The table shows that a device profile may be either publicly defined or vendor specific:

**Table 6-6.1 Device Profile Numbering**

Type	Range	Quantity
Publicly Defined	00 <sub>hex</sub> - 63 <sub>hex</sub>	100
Vendor Specific	64 <sub>hex</sub> - C7 <sub>hex</sub>	100
Reserved by CIP	C8 <sub>hex</sub> - FF <sub>hex</sub>	56
Publicly Defined	100 <sub>hex</sub> - 2FF <sub>hex</sub>	512
Vendor Specific	300 <sub>hex</sub> - 4FF <sub>hex</sub>	512
Reserved by CIP	500 <sub>hex</sub> - FFFF <sub>hex</sub>	64,256

While you are highly encouraged to adopt or develop a device profile for your product you may be unwilling or unable to do so. For this reason ranges of device type numbers have been set aside for "Vendor Specific" device profiles. If you choose to use one of these device type numbers you are not required to publish a device profile for your product. It is important to note, however, that if you do not publish your device's profile, your customers will not be able to find direct replacements for your product and, more importantly, they will not be able to use your product as a direct replacement for your competitor's product. Additionally, even vendor specific device profiles are required to support the minimum objects listed in section 6-2.1.

## 6-7 Device Profiles

The remainder of this chapter contains listings of all existing device profiles at the time of publication.

**Table 6-7.1 Device Profiles in This Chapter**

For information about:	Go to page:	Go to section:	Device Type Number:
AC Drives	6-56	6-15	02 <sub>hex</sub>
Barcode Scanner	6-70	6-17	Not yet assigned
Modbus Device	See Volume 7, Integration of Modbus Devices in the CIP Architecture		28 <sub>hex</sub>
CIP Motion Drive	6-249	6-41	25 <sub>hex</sub>
Circuit Breaker	6-115	6-25	Not yet assigned
Communications Adapter	6-53	6-13	0C <sub>hex</sub>
CompoNet Repeater	See Volume 6, CompoNet Adaptation of CIP		26 <sub>hex</sub>
Contactor	6-128	6-27	15 <sub>hex</sub>
Control Station	6-113	6-23	Not yet assigned
ControlNet Physical Layer Component	6-181	6-34	32 <sub>hex</sub>
Programmable Logic Controller	6-180	6-33	0E <sub>hex</sub>
DC Drives	6-56	6-15	13 <sub>hex</sub>
DC Power Generator	6-214	6-38	1F <sub>hex</sub>
Encoder	6-101	6-21	22 <sub>hex</sub>
Fluid Flow Controller	6-240	6-40	24 <sub>hex</sub>
General Purpose Analog I/O	6-55	6-14	Not yet assigned
General Purpose Discrete I/O	6-37	6-12	07 <sub>hex</sub>
Generic Device	6-23	6-8	00 <sub>hex</sub>
Human-Machine Interface	6-149	6-30	18 <sub>hex</sub>
Inductive Proximity Switch	6-29	6-10	05 <sub>hex</sub>
Limit Switch	6-25	6-9	04 <sub>hex</sub>
Mass Flow Controller	6-151	6-31	1A <sub>hex</sub>
Mass Flow Controller, Enhanced	6-292	6-42	27 <sub>hex</sub>
Message Display	6-114	6-24	Not yet assigned
Motor Overload	6-94	6-19	03 <sub>hex</sub>
Motor Starter	6-133	6-28	16 <sub>hex</sub>
Photoelectric Sensor	6-33	6-11	06 <sub>hex</sub>
Pneumatic Valve(s)	6-116	6-26	1B <sub>hex</sub>
Position Controller	6-71	6-18	10 <sub>hex</sub>
Process Control Valve	6-182	6-35	1D <sub>hex</sub>
Residual Gas Analyzer	6-206	6-37	1E <sub>hex</sub>
Resolver	6-106	6-22	09 <sub>hex</sub>
RF Power Generator	6-227	6-39	20 <sub>hex</sub>
Safety Discrete I/O Device	See Volume 5, CIP Safety		23 <sub>hex</sub>

For information about:	Go to page:	Go to section:	Device Type Number:
Servo Drives	6-69	6-16	Not yet assigned
Softstart Starter	6-142	6-29	17 <sub>hex</sub>
Turbomolecular Vacuum Pump	6-194	6-36	21 <sub>hex</sub>
Weigh Scale	6-100	6-20	Not yet assigned
Vacuum Pressure Gauge	6-163	6-32	1C <sub>hex</sub>

The following device type numbers have been obsoleted.

**Table 6-7.2 Obsolete Device Profiles**

Obsolete Device Type Number	Previous Profile Assignment
01 <sub>hex</sub>	Control Station
08 <sub>hex</sub>	Encoder
0A <sub>hex</sub>	General Purpose Analog I/O
0D <sub>hex</sub>	Barcode Scanner
11 <sub>hex</sub>	Weigh Scale
12 <sub>hex</sub>	Message Display
14 <sub>hex</sub>	Servo Drives
19 <sub>hex</sub>	Pneumatic Valve(s)

## 6-8 Generic Device

### Device Type: 00 Hex

The Generic Device type defines a device that does not fit into any of the defined device types. Initially, there will probably be many Generic Device type devices, but over time, Open DeviceNet Vendor Association, Inc. and ControlNet International Special Interest Groups (SIGs) will create a specific device profile for devices with similar functionality. The Generic Device type devices are not interchangeable.

#### 6-8.1 Object Model

The Object Model in Figure 6-8.1 represents the minimum support in a Generic Device. The table below indicates:

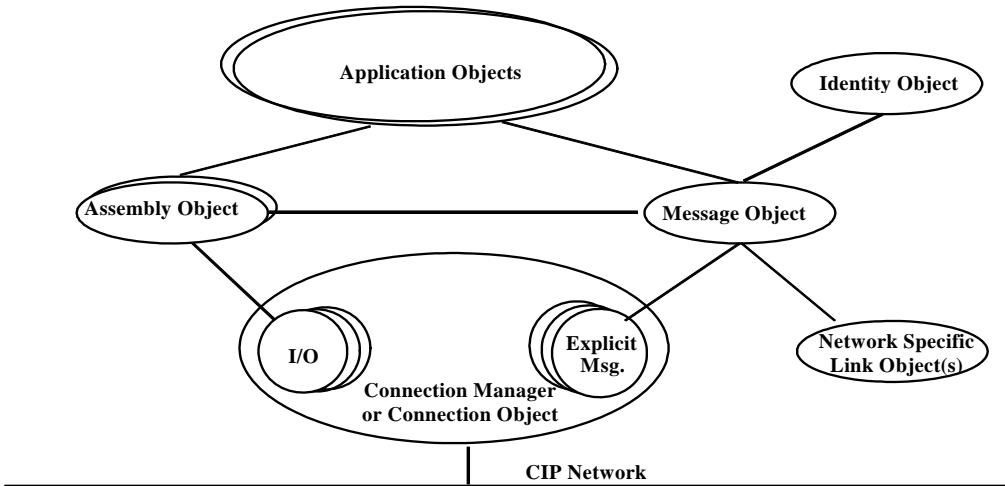
- the object classes present in this device
- whether or not the class is required
- the number of instances present in each class

**Table 6-8.1 Objects Present in a Generic Device**

Object Class	Optional/Required	# of Instances
CIP Common Required Objects	Required	See Section 6-2.1
Assembly	Required	at least 1
Application	Required	at least 1

The Generic Device profile does not specify any instances of the Assembly Object or the type of application objects necessary for device operation. This portion of the device profile must be supplied by the product developer as described in Section 6-1 through 6-7 inclusive. The Generic device profile shall be unique among the “open” device profiles in permitting the use of Assembly object instances in the open range.

**Figure 6-8.2 Object Model for the Generic Device**



## 6-8.2 How Objects Affect Behavior

The objects for this device affect the device's behavior as shown in the table below.

**Table 6-8.3 Object Effect on Behavior**

Object	Effect on behavior
CIP Common Required	See Section 6-2.2 for details.
Assembly	Defines input/output and configuration data format
Application	Defines device operation

## 6-8.3 Defining Object Interfaces

The objects in the Generic Device have the interfaces listed in the following table:

**Table 6-8.4 Object Interfaces**

Object	Interface
CIP Common Required	See Section 6-2.3 for details.
Assembly	I/O Connection or Message Router
Application	Assembly or Message Router

## 6-9 Limit Switch Device

### Device Type: 04 Hex

A limit switch mechanically detects the presence or absence of a physical target object. The switch detects an object when a lever or rod makes physical contact with the object.

#### 6-9.1 Object Model

The Object Model in Figure 6-9.2. represents a limit switch. The table below indicates:

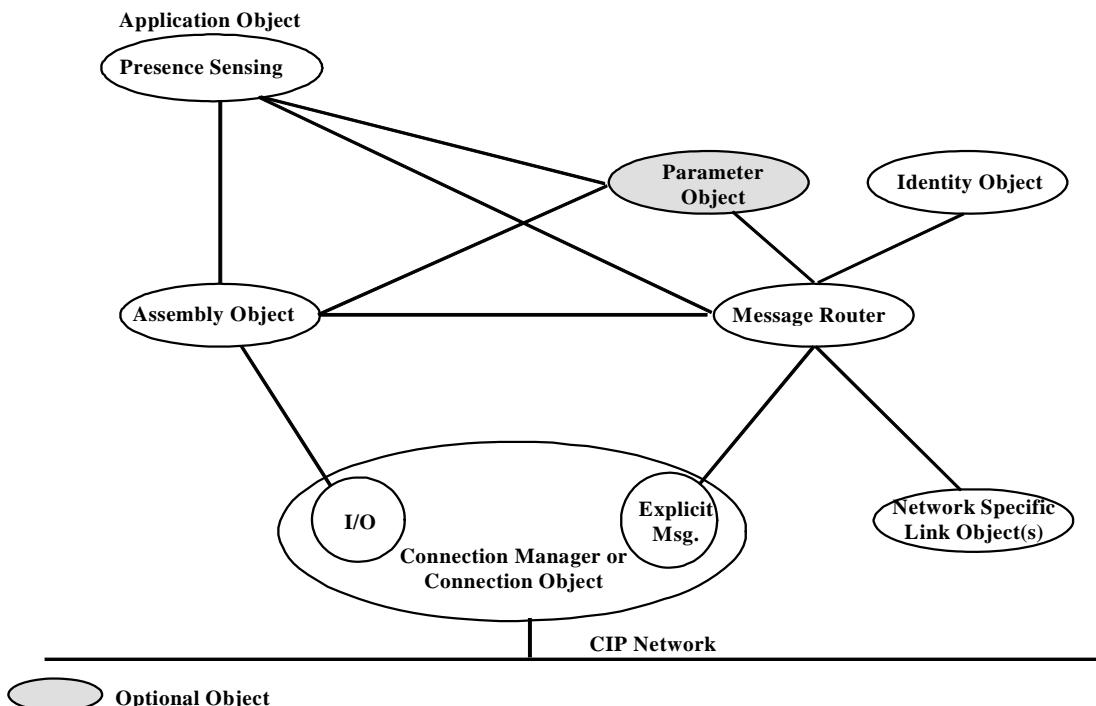
- the object classes present in this device
- whether or not the class is required
- the number of instances present in each class

The CIP Object Library provides more details about these objects.

**Table 6-9.1 Objects Present in a Limit Switch**

Object Class	Optional/Required	# of Instances
CIP Common Required Objects	Required	See Section 6-2.1
Assembly	Required	1
Parameter	Optional	1
Presence Sensing	Required	1

**Figure 6-9.2 Object Model for a Limit Switch**



## 6-9.2 How Objects Affect Behavior

The objects for this device affect the device's behavior as shown in the table below.

**Table 6-9.3 Object Effect on Behavior**

Object	Effect on behavior
CIP Common Required	See Section 6-2.2 for details.
Assembly	Defines I/O data format
Parameter	Provides a public interface to the device's configuration data
Presence Sensing	Affects <i>Output Value</i> (attribute)

## 6-9.3 Defining Object Interfaces

The objects in this device have the interfaces listed in the following table:

**Table 6-9.4 Object Interfaces**

Object	Interface
CIP Common Required	See Section 6-2.3 for details.
Assembly	I/O Connection or Message Router
Parameter	Message Router
Presence Sensing	Message Router, Assembly Object, or Parameter Object

## 6-9.4 I/O Assembly Instances

The following table identifies the I/O assembly instance supported by the limit switch.

**Table 6-9.5 I/O Assembly Instances**

Number	Type	Name
1	Input	Input Data

## 6-9.5 I/O Assembly Data Attribute Format

The I/O Assembly data attribute has the format shown below.

**Table 6-9.6 I/O Assembly Data Attribute Format**

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	0	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Diagnostic	Output

## 6-9.6 Mapping I/O Assembly Data Attribute Components

The following table indicates the I/O assembly Data attribute mapping for this limit switch device.

**Table 6-9.7 I/O Assembly Data Mapping**

Data Component Name	Class		Instance Number	Attribute	
	Name	Number		Name	Number
Diagnostic	presence sensing	0E <sub>hex</sub>	1	Diagnostic	4
Output	presence sensing	0E <sub>hex</sub>	1	Output	1

## 6-9.7 Defining Device Configuration

Public access to the Presence Sensing Object by the Message Router must be supported for configuration of this device type. If supported, the optional Parameter Object may be used to access the device type's configuration parameter.

### 6-9.7.1 Parameter Object Instances

The limit switch contains one instance of the Parameter Object Class. This instance is a Parameter Object stub. See The CIP Object Library for the definition of the Parameter Object and an explanation of how it is used for configuration.

The following table identifies the Parameter Object instance supported by the limit switch.

**Table 6-9.8 Parameter Object Instances Supported**

Number	Name
1	Operation Mode Configuration

### 6-9.7.2 Mapping Parameter Object Data

The following table indicates the Parameter Object data mapping for the limit switch device.

**Table 6-9.9 Parameter Object Data Mapping**

Configuration Parameter Name	Class		Instance Number	Attribute	
	Name	Number		Name	Number
Operate Mode Configuration	presence sensing	0E <sub>hex</sub>	1	Operate Mode	8

### 6-9.7.3 Configuration Parameter Definitions

The following sections of an example EDS show the information necessary to define the configuration parameters for a limit switch.

```
[Params]
Param1= $Operate Mode
0, $Data Placeholder
2,"20 0e 24 01 30 08", $Path size and Path to Operate Mode Attr
0x0002, $Descriptor (support enumerated strings
0xC1,1 $Data Type and Size (Boolean)
"Operate Mode", $Name
", $Units (not used)
", $User Manual Ref (not used)
0,1,0, $min, max, default values
0,0,0,0, $mult, div, base, offset scaling (not used)
0,0,0,0, $mult, div, base, offset links (not used)
1; $decimal places

[EnumPar]
Param1= $Operate Mode Enumerated Strings
"Normally Open", $For value=0
"Normally Closed"; $For value=1
```

### 6-9.8 Effect of Configuration Parameters on Behavior

The configuration parameter affects the device's behavior as shown below.

**Table 6-9.10 Configuration Parameter Effect on Behavior**

Parameter	Effect on Behavior
Operate Mode	Inverts the level defined for the Output attribute of the Presence Sensing Object

## 6-10 Inductive Proximity Switch Device

### Device Type: 05 Hex

An inductive proximity switch operates in an electromagnetic field. When it senses a change in the field, it sends a signal to an output amplifier circuit to change the state of the circuit.

### 6-10.1 Object Model

The Object Model in Figure 6-10.2 represents an inductive proximity switch. The table below indicates:

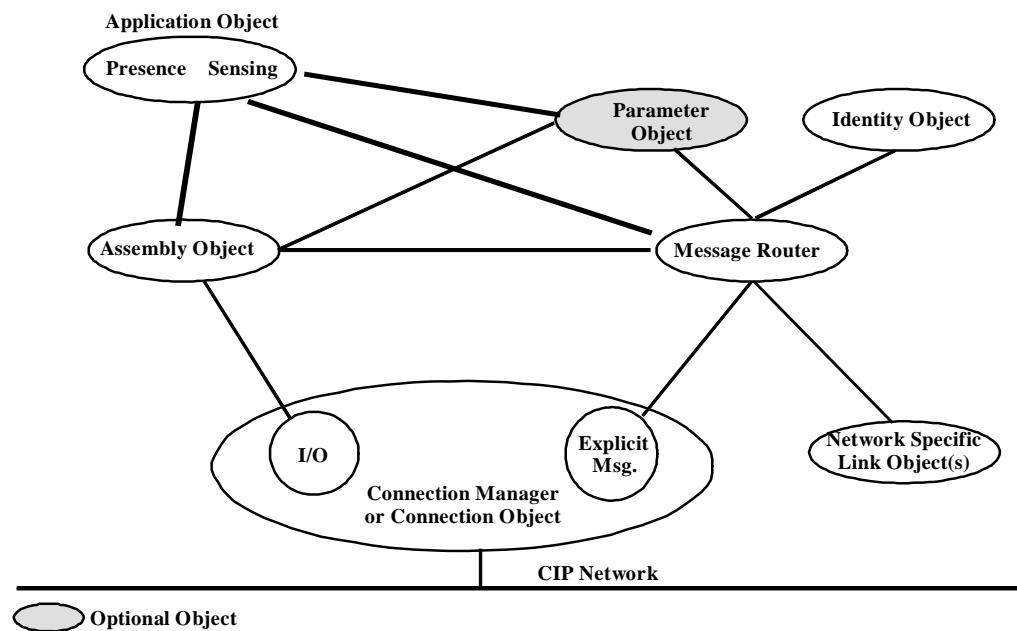
- the object classes present in this device
- whether or not the class is required
- the number of instances present in each class

The CIP Object Library provides more details about these objects.

**Table 6-10.1 Objects Present in an Inductive Proximity Switch**

Object Class	Optional/Required	# of Instances
CIP Common Required Objects	Required	See Section 6-2.1
Assembly	Required	1
Parameter	Optional	1
Presence Sensing	Required	1

**Figure 6-10.2 Object Model for an Inductive Proximity Switch**



## 6-10.2 How Objects Affect Behavior

The objects for this device affect the device's behavior as shown in the table below.

**Table 6-10.3 Object Effect on Behavior**

Object	Effect on behavior
CIP Common Required	See Section 6-2.2 for details.
Assembly	Defines I/O data format
Parameter	Provides a public interface to the device's configuration data
Presence Sensing	Effects <i>Output Value</i> (attribute)

## 6-10.3 Defining Object Interfaces

The objects in this device have the interfaces listed in the following table:

**Table 6-10.4 Object Interfaces**

Object	Interface
CIP Common Required	See Section 6-2.3 for details.
Assembly	I/O Connection or Message Router
Parameter	Message Router
Presence Sensing	Message Router, Assembly Object, or Parameter Object

## 6-10.4 I/O Assembly Instances

The following table identifies the I/O assembly instance supported by the inductive proximity switch.

**Table 6-10.5 I/O Assembly Instances**

Number	Type	Name
1	Input	Input Data

## 6-10.5 I/O Assembly Data Attribute Format

The I/O Assembly data attribute has the format shown below.

**Table 6-10.6 I/O Assembly Data Attribute Format**

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	0	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Diagnostic	Output

## 6-10.6 Mapping I/O Assembly Data Attribute Components

The following table indicates the I/O assembly Data attribute mapping for this inductive proximity switch device.

**Table 6-10.7 I/O Assembly Data Mapping**

Data Component Name	Class		Instance Number	Attribute	
	Name	Number		Name	Number
Diagnostic	Presence Sensing	0E <sub>hex</sub>	1	Diagnostic	4
Output	Presence Sensing	0E <sub>hex</sub>	1	Output	1

## 6-10.7 Defining Device Configuration

Public access to the Presence Sensing Object by the Message Router must be supported for configuration of this device type. If supported, the optional Parameter Object may be used to access the device type's configuration parameter.

### 6-10.7.1 Parameter Object Instances

The following table identifies the Parameter Object instance supported by the inductive proximity switch.

**Table 6-10.8 Parameter Object Instances Supported**

Number	Name
1	Operation Mode Configuration

### 6-10.7.2 Mapping Parameter Object Data

The following table indicates the Parameter Object data mapping for the inductive proximity switch device.

**Table 6-10.9 Parameter Object Data Mapping**

Configuration Parameter Name	Class		Instance Number	Attribute	
	Name	Number		Name	Number
Operate Mode Configuration	Presence Sensing	0E <sub>hex</sub>	1	Operate Mode	8

### 6-10.7.3 Configuration Parameter Definitions

The following sections of an example EDS show the information necessary to define the configuration parameters for an inductive proximity switch.

```
[Parms]
Param1= $Operate Mode
0,          $Data Placeholder
3,"20 0e 24 01 30 08", $Path size and Path to Operate Mode Attr
0x0002,      $Descriptor (support enumerated strings)
0xC1,1       $Data Type and Size (Boolean)
"Operate Mode", $Name
",",          $Units (not used)
",",          $User Manual Ref (not used)
0,1,0,        $min, max, default values
0,0,0,0,      $mult, div, base, offset scaling (not used)
0,0,0,0,      $mult, div, base, offset links (not used)
1;           $decimal places

[EnumPar]
Param1= $Operate Mode Enumerated Strings
"Normally Open", $For value=0
"Normally Closed"; $For value=1
```

### 6-10.8 Effect of Configuration Parameters on Behavior

The configuration parameter affects the device's behavior as shown below.

**Table 6-10.10 Configuration Parameter Effect on Behavior**

Parameter	Effect on Behavior
Operate Mode	Inverts the level defined for the Output attribute of the Presence Sensing Object

## 6-11 Photoelectric Sensor Device

### Device Type: 06 Hex

A photoelectric sensor electrically senses the presence or absence of a target object or part of a machine. Typical applications include assembly, packaging, and material handling.

#### 6-11.1 Object Model

The Object Model in Figure 6-11.2 represents a photoelectric sensor. The table below indicates:

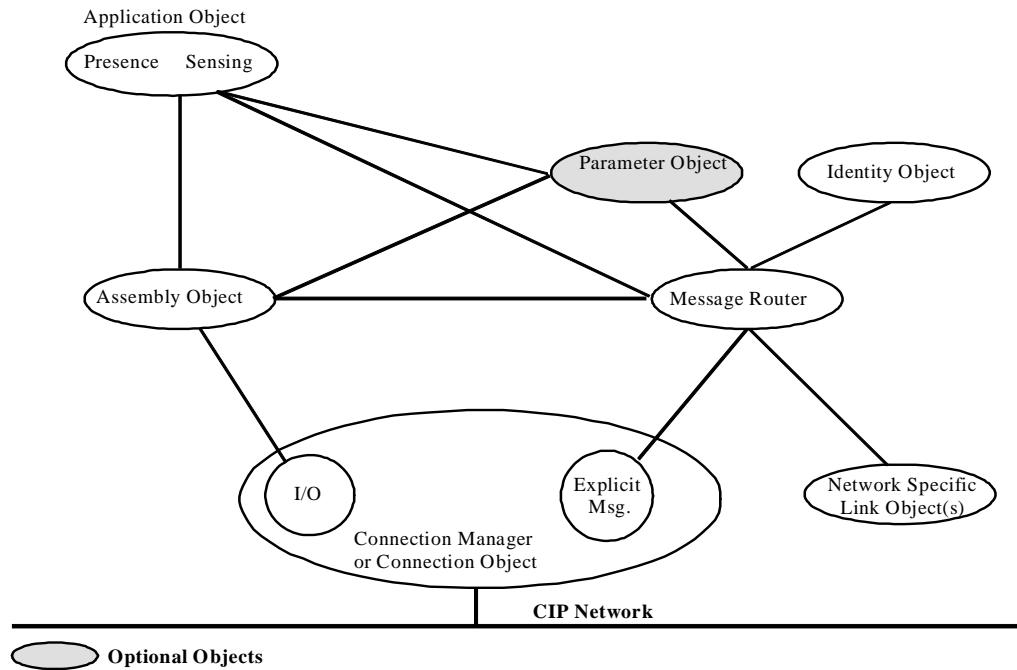
- the object classes present in this device
- whether or not the class is required
- the number of instances present in each class

The CIP Object Library provides more details about these objects.

**Table 6-11.1 I/O Objects Present in a Photoelectric Sensor**

Object Class	Optional/Required	# of Instances
CIP Common Required Objects	Required	See Section 6-2.1
Assembly	Required	1
Parameter	Optional	1
Presence Sensing	Required	1

**Figure 6-11.2 Object Model for a Photoelectric Sensor**



## 6-11.2 How Objects Affect Behavior

The objects for this device affect the device's behavior as shown in the table below.

**Table 6-11.3 Object Effect on Behavior**

Object	Effect on behavior
CIP Common Required	See Section 6-2.2 for details.
Assembly	Defines I/O data format
Parameter	Provides a public interface to the device's configuration data
Presence Sensing	Affects output value

## 6-11.3 Defining Object Interfaces

The objects in this device have the interfaces listed in the following table:

**Table 6-11.4 Object Interfaces**

Object	Interface
CIP Common Required	See Section 6-2.3 for details.
Assembly	I/O Connection or Message Router
Parameter	Message Router
Presence Sensing	Message Router, Assembly Object, or Parameter Object

## 6-11.4 I/O Assembly Instances

The following table identifies the I/O assembly instance supported by the photoelectric sensor.

**Table 6-11.5 I/O Assembly Instances**

Number	Type	Name
1	Input	Input Data

## 6-11.5 I/O Assembly Data Attribute Format

The I/O Assembly data attribute has the format shown below.

**Table 6-11.6 I/O Assembly Data Attribute Format**

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	0	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Diagnostic	Output

## 6-11.6 Mapping I/O Assembly Data Attribute Components

The following table indicates the I/O assembly Data attribute mapping for this photoelectric sensor device.

**Table 6-11.7 I/O Assembly Data Mapping**

Data Component Name	Class		Instance Number	Attribute	
	Name	Number		Name	Number
Diagnostic	Presence Sensing	0E <sub>hex</sub>	1	Diagnostic	4
Output	Presence Sensing	0E <sub>hex</sub>	1	Output	1

## 6-11.7 Defining Device Configuration

Public access to the Presence-Sensing Object by the Message Router must be supported for configuration of this device type. If supported, the optional Parameter Object may be used to access the device type's configuration parameter.

### 6-11.7.1 Parameter Object Instances

The photoelectric sensor contains one instance of the Parameter Object Class. This instance is a Parameter Object stub. See Chapter 5, Object Library, for the definition of the Parameter Object and an explanation of how it is used for configuration.

The following table identifies the Parameter Object instance supported by the photoelectric sensor.

**Table 6-11.8 Parameter Object Instances Supported**

Number	Name
1	Operation Mode Configuration

### 6-11.7.2 Mapping Parameter Object Data

The following table indicates the Parameter Object data mapping for the photoelectric sensor device.

**Table 6-11.9 Parameter Object Data Mapping**

Configuration Parameter Name	Class		Instance Number	Attribute	
	Name	Number		Name	Number
Operate Mode Configuration	Presence Sensing	0E <sub>hex</sub>	1	Operate Mode	8

### 6-11.7.3 Configuration Parameter Definitions

The following sections of an example EDS show the information necessary to define the configuration parameters for photoelectric sensor.

```
[Parms]
Param1= $Operate Mode
0,           $Data Placeholder
2,"20 0e 24 01 30 08", $Path size and Path to Operate Mode Attr
0x0002,      $Descriptor (support enumerated strings
0xC1,1       $Data Type and Size (Boolean)
"Operate Mode", $Name
",",          $Units (not used)
",",          $User Manual Ref (not used)
0,1,0,        $min, max, default values
0,0,0,0,      $mult, div, base, offset scaling (not used)
0,0,0,0,      $mult, div, base, offset links (not used)
1;            $decimal places
[EnumPar]
Param1= $Operate Mode Enumerated Strings
"Light Operate", $For value=0
"Dark Operate"; $For value=1
```

### 6-11.8 Effect of Configuration Parameters on Behavior

The configuration parameter affects the device's behavior as shown below.

**Table 6-11.10 Configuration Parameter Effect on Behavior**

Parameter	Effect on Behavior
Operate Mode	Inverts the level defined for the Output attribute of the Presence Sensing Object

## 6-12 General Purpose Discrete I/O Device

### Device Type: 07 Hex

A General Purpose Discrete I/O device type interfaces to multiple discrete I/O device types that do not have network capabilities. Examples include sensors and actuators.

#### 6-12.1 Object Model

The Object Model in Figure 6-12.2 represents a General Purpose Discrete I/O device. The table below indicates:

- the object classes present in this device
- whether or not the class is required
- the number of instances present in each class

The CIP Object Library provides more details about these objects.

**Table 6-12.1 Objects Present in the General Purpose Discrete I/O Device**

Object Class	Optional/Required	# of Instances
CIP Common Required Objects	Required	See Section 6-2.1
Assembly	Required	*
Discrete Input Group	Optional	1
Discrete Output Group	Optional	1
Discrete Input Point	**	*
Discrete Output Point	***	*

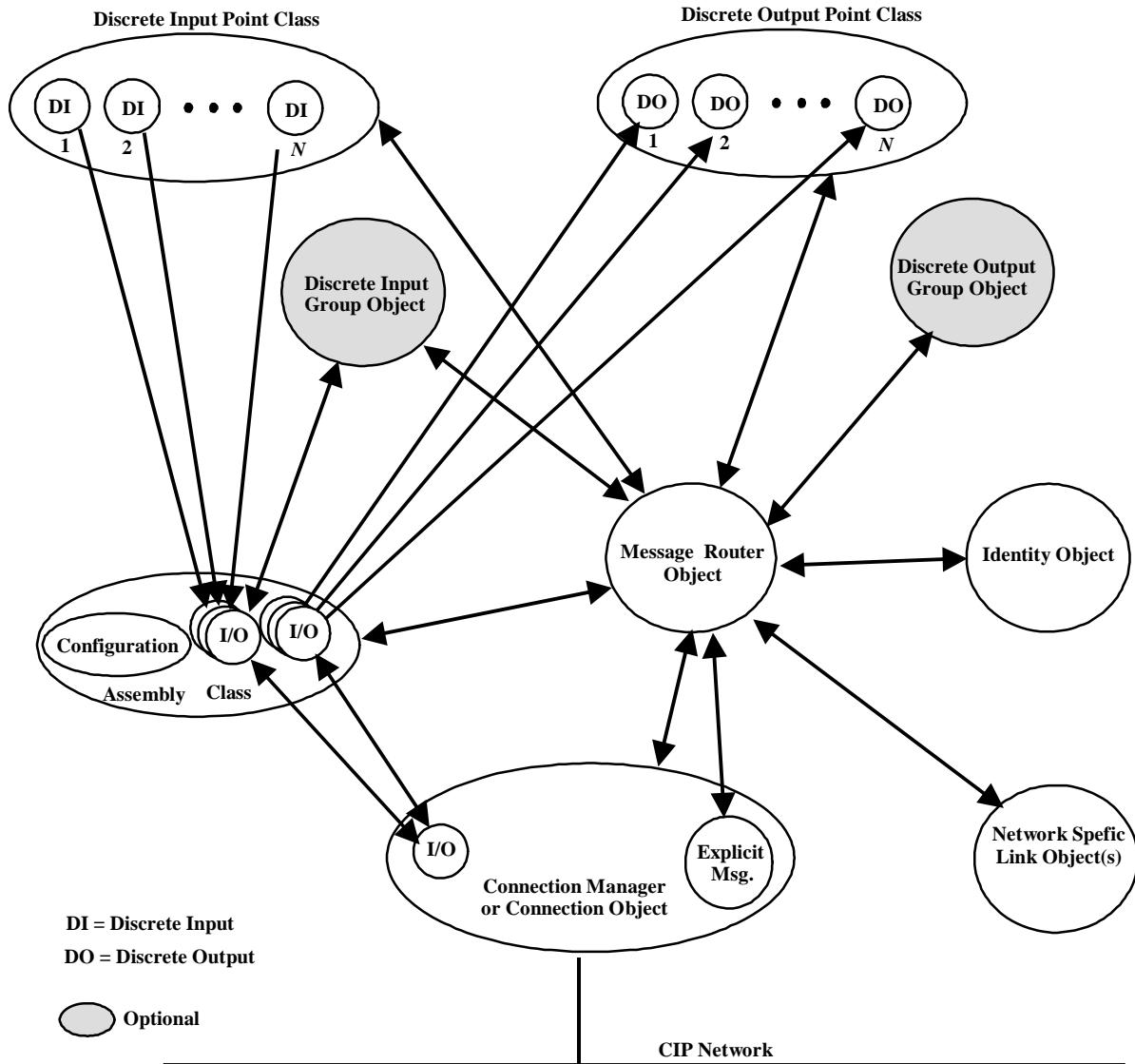
\* Depends on the level of I/O support provided by the product.

\*\* Required for input functions

\*\*\* Required for output functions

General Purpose Discrete I/O Device, Type: 07<sub>Hex</sub>

Figure 6-12.2 Object Model for a General Purpose Discrete I/O Device



## 6-12.2 How Objects Affect Behavior

The objects for this device affect the device's behavior as shown in the table below.

**Table 6-12.3 Object Effect on Behavior**

Object	Effect on behavior
CIP Common Required	See Section 6-2.2 for details.
Assembly	Defines I/O data format and Output Configuration data format
Discrete Input Point	Defines behavior of the discrete input points for this device
Discrete Input Group	Stores the combined status of the Discrete Input Points
Discrete Output Point	Defines the behavior of discrete output points for this device
Discrete Output Group	Defines the Idle and Fault actions of the discrete output points

## 6-12.3 Defining Object Interfaces

The objects in this device have the interfaces listed in the following table:

**Table 6-12.4 Object Interfaces**

Object	Interface
CIP Common Required	See Section 6-2.3 for details.
Assembly	I/O Connection or Message Router
Discrete Input Group	Message Router
Discrete Output Group	Message Router
Discrete Input Point	Message Router
Discrete Output Point	Message Router or Assembly Object

## 6-12.4 I/O Assembly Instances

The General Purpose Discrete I/O device I/O assemblies consist of:

- six predefined input assemblies with single input status bits
- one product-specific input assembly with a single input status bit
- six predefined input assemblies with multiple input status bits
- one product-specific input assembly with multiple input status bits
- six predefined output assemblies
- one product-specific output assembly
- six predefined input assemblies with output status bits
- one product-specific output status assembly
- four input assemblies with multiple input status bits and multiple output status bits
- nine input assemblies with a single input status bit and multiple output status bits

**General Purpose Discrete I/O Device, Type: 07<sub>Hex</sub>**

The following table identifies the I/O assembly instances supported by this device.

**Table 6-12.5 I/O Assembly Instances**

<b>Number</b>	<b>Type</b>	<b>Name</b>
1	Input	1-Point Input with No Status Bit
2	Input	2-Point Input with No Status Bit
3	Input	4-Point Input with No Status Bit
4	Input	8-Point Input with No Status Bit
5	Input	16-Point Input with No Status Bit
6	Input	32-Point Input with No Status Bit
7	Input	N-Point Input with No Status Bit
11	Input	1-Point Input with Single Status Bits
12	Input	2-Point Input with Single Status Bit
13	Input	4-Point Input with Single Status Bit
14	Input	8-Point Input with Single Status Bit
15	Input	16-Point Input with Single Status Bit
16	Input	32-Point Input with Single Status Bit
17	Input	N-Point Input with Single Status Bit
21	Input	1-Point Input with Multiple Status Bits
22	Input	2-Point Input with Multiple Status Bits
23	Input	4-Point Input with Multiple Status Bits
24	Input	8-Point Input with Multiple Status Bits
25	Input	16-Point Input with Multiple Status Bits
26	Input	32-Point Input with Multiple Status Bits
27	Input	N-Point Input with Multiple Status Bits
31	Output	1-Point Output
32	Output	2-Point Output
33	Output	4-Point Output
34	Output	8-Point Output
35	Output	16-Point Output
36	Output	32-Point Output
37	Output	N-Point Output
41	Input	1-Point Output Status Bit
42	Input	2-Point Output Status Bits
43	Input	4-Point Output Status Bits
44	Input	8-Point Output Status Bits
45	Input	16-Point Output Status Bits
46	Input	32-Point Output Status Bits
47	Input	N-Point Output Status Bits

**General Purpose Discrete I/O Device, Type: 07<sub>Hex</sub>**

<b>Number</b>	<b>Type</b>	<b>Name</b>
52	Input	2-Point Input with Single Input Status and Single Output Status Bits
53	Input	4-Point Input with Single Input Status and Single Output Status Bits
54	Input	8-Point Input with Single Input Status and Single Output Status Bits
55	Input	16-Point Input with Single Input Status and Single Output Status Bits
56	Input	32-Point Input with Single Input Status and Single Output Status Bits
57	Input	N-Point Input with Single Input Status and Single Output Status Bits
62	Input	2-Point Input with Multiple Input Status and Multiple Output Status Bits
63	Input	4-Point Input with Multiple Input Status and Multiple Output Status Bits
64	Input	8-Point Input with Multiple Input Status and Multiple Output Status Bits
65	Input	16-Point Input with Multiple Input Status and Multiple Output Status Bits
70	Input	1-Point Input with Single Input Status and 1 Output Status Bit
71	Input	2-Point Input with Single Input Status and 1 Output Status Bit
72	Input	2-Point Input with Single Input Status and 2 Output Status Bits
73	Input	4-Point Input with Single Input Status and 2 Output Status Bits
74	Input	4-Point Input with Single Input Status and 4 Output Status Bits
75	Input	8-Point Input with Single Input Status and 4 Output Status Bits
76	Input	8-Point Input with Single Input Status and 8 Output Status Bits
77	Input	16-Point Input with Single Input Status and 8 Output Status Bits
78	Input	16-Point Input with Single Input Status and 16 Output Status Bits

**6-12.5 I/O Assembly Data Attribute Format**

The I/O Assembly data attribute for the input data with no status bit has the format shown below.

**Table 6-12.6 I/O Assembly Data Attribute Format – Instances w/no Status Bit**

<b>Instance</b>	<b>Byte</b>	<b>Bit 7</b>	<b>Bit 6</b>	<b>Bit 5</b>	<b>Bit 4</b>	<b>Bit 3</b>	<b>Bit 2</b>	<b>Bit 1</b>	<b>Bit 0</b>
1	0	Reserved							Discrete Input1
2	0	Reserved						Discrete Input2	Discrete Input 1
3	0	Reserved			Discrete Input4	Discrete Input3	Discrete Input2	Discrete Input1	
4	0	Discrete Input8	Discrete Input7	Discrete Input6	Discrete Input5	Discrete Input4	Discrete Input3	Discrete Input2	Discrete Input1

General Purpose Discrete I/O Device, Type: 07<sub>Hex</sub>

5	0	Discrete Input8	Discrete Input7	Discrete Input6	Discrete Input5	Discrete Input4	Discrete Input3	Discrete Input2	Discrete Input1
	1	Discrete Input16	Discrete Input15	Discrete Input14	Discrete Input13	Discrete Input12	Discrete Input11	Discrete Input10	Discrete Input9
6	0	Discrete Input8	Discrete Input7	Discrete Input6	Discrete Input5	Discrete Input4	Discrete Input3	Discrete Input2	Discrete Input1
	1	Discrete Input16	Discrete Input15	Discrete Input14	Discrete Input13	Discrete Input12	Discrete Input11	Discrete Input10	Discrete Input9
	2	Discrete Input24	Discrete Input23	Discrete Input22	Discrete Input21	Discrete Input20	Discrete Input19	Discrete Input18	Discrete Input17
	3	Discrete Input32	Discrete Input31	Discrete Input30	Discrete Input29	Discrete Input28	Discrete Input27	Discrete Input26	Discrete Input25
7	0	Discrete Input8	Discrete Input7	Discrete Input6	Discrete Input5	Discrete Input4	Discrete Input3	Discrete Input2	Discrete Input1
	•								
	M	reserved						Discrete Input N	Discrete Input N-1

The I/O Assembly data attribute for the input data with one status bit has the format shown below.

Table 6-12.7 I/O Assembly Data Attribute Format – Instances w/one Status Bit

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
11	0	Status	Reserved					Discrete Input1	
12	0	Status	Reserved					Discrete Input2	Discrete Input1
13	0	Status	Reserved			Discrete Input4	Discrete Input3	Discrete Input2	Discrete Input1
14	0	Discrete Input8	Discrete Input7	Discrete Input6	Discrete Input5	Discrete Input4	Discrete Input3	Discrete Input2	Discrete Input1
	1	Status	Reserved						
	2	Discrete Input8	Discrete Input7	Discrete Input6	Discrete Input5	Discrete Input4	Discrete Input3	Discrete Input2	Discrete Input1
	1	Discrete Input16	Discrete Input15	Discrete Input14	Discrete Input13	Discrete Input12	Discrete Input11	Discrete Input10	Discrete Input9

General Purpose Discrete I/O Device, Type: 07<sub>Hex</sub>

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
16	0	Discrete Input8	Discrete Input7	Discrete Input6	Discrete Input5	Discrete Input4	Discrete Input3	Discrete Input2	Discrete Input1
	1	Discrete Input16	Discrete Input15	Discrete Input14	Discrete Input13	Discrete Input12	Discrete Input11	Discrete Input10	Discrete Input9
	2	Discrete Input24	Discrete Input23	Discrete Input22	Discrete Input21	Discrete Input20	Discrete Input19	Discrete Input18	Discrete Input17
	3	Discrete Input32	Discrete Input31	Discrete Input30	Discrete Input29	Discrete Input28	Discrete Input27	Discrete Input26	Discrete Input25
	4	Status	Reserved						
17	0	Discrete Input8	Discrete Input7	Discrete Input6	Discrete Input5	Discrete Input4	Discrete Input3	Discrete Input2	Discrete Input1
	•								
	M	Status	Reserved				Discrete Input N	Discrete Input N-1	Discrete Input N-2

The I/O Assembly data attribute for the input data with multiple status bits has the format shown below.

Table 6-12.8 I/O Assembly Data Attribute Format – Instances w/multiple Status Bits

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
22	0	Reserved				Status2	Status1	Discrete Input2	Discrete Input1
23	0	Status4	Status3	Status2	Status1	Discrete Input4	Discrete Input3	Discrete Input2	Discrete Input1
24	0	Discrete Input8	Discrete Input7	Discrete Input6	Discrete Input5	Discrete Input4	Discrete Input3	Discrete Input2	Discrete Input1
	1	Status8	Status7	Status6	Status5	Status4	Status3	Status2	Status1
25	0	Discrete Input8	Discrete Input7	Discrete Input6	Discrete Input5	Discrete Input4	Discrete Input3	Discrete Input2	Discrete Input1
	1	Discrete Input16	Discrete Input15	Discrete Input14	Discrete Input13	Discrete Input12	Discrete Input11	Discrete Input10	Discrete Input9
	2	Status8	Status7	Status6	Status5	Status4	Status3	Status2	Status1
	3	Status16	Status15	Status14	Status13	Status12	Status11	Status10	Status9

General Purpose Discrete I/O Device, Type: 07<sub>Hex</sub>

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
26	0	Discrete Input8	Discrete Input7	Discrete Input6	Discrete Input5	Discrete Input4	Discrete Input3	Discrete Input2	Discrete Input1
	•								
	•								
	•								
	3	Discrete Input32	•	•	•	•	•	•	Discrete Input25
	4	Status8	•	•	•	•	•	•	Status1
	•								
	•								
	7	Status32	•	•	•	•	•	•	Status25
27	0	Discrete Input8	Discrete Input7	Discrete Input6	Discrete Input5	Discrete Input4	Discrete Input3	Discrete Input2	Discrete Input1
	•								
	•								
	•								
	M-2	Status5	Status4	Status3	Status2	Status1	Discrete Input N	Discrete Input N-1	Discrete Input N-2
M-1	•	•	•	•	•	•	•	•	Status6
	M	Reserved					Status N	Status N-1	Status N-2

The I/O Assembly data attribute for the output data has the format shown below.

Table 6-12.9 I/O Assembly Data Attribute Format – Output Data Assembly

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
31	0	Reserved						Discrete Output1	
32	0	Reserved						Discrete Output2	Discrete Output1
33	0	Reserved				Discrete Output4	Discrete Output3	Discrete Output2	Discrete Output1
34	0	Discrete Output8	Discrete Output7	Discrete Output6	Discrete Output5	Discrete Output4	Discrete Output3	Discrete Output2	Discrete Output1
35	0	Discrete Output8	Discrete Output7	Discrete Output6	Discrete Output5	Discrete Output4	Discrete Output3	Discrete Output2	Discrete Output1
	1	Discrete Output16	Discrete Output15	Discrete Output14	Discrete Output13	Discrete Output12	Discrete Output11	Discrete Output10	Discrete Output9

General Purpose Discrete I/O Device, Type: 07<sub>Hex</sub>

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
36	0	Discrete Output8	Discrete Output7	Discrete Output6	Discrete Output5	Discrete Output4	Discrete Output3	Discrete Output2	Discrete Output1
	1	Discrete Output16	Discrete Output15	Discrete Output14	Discrete Output13	Discrete Output12	Discrete Output11	Discrete Output10	Discrete Output9
	2	Discrete Output24	Discrete Output23	Discrete Output22	Discrete Output21	Discrete Output20	Discrete Output19	Discrete Output18	Discrete Output17
	3	Discrete Output32	Discrete Output31	Discrete Output30	Discrete Output29	Discrete Output28	Discrete Output27	Discrete Output26	Discrete Output25
37	0	Discrete Output8	Discrete Output7	Discrete Output6	Discrete Output5	Discrete Output4	Discrete Output3	Discrete Output2	Discrete Output1
	•								
	M	reserved					Discrete Output N	Discrete Output N-1	Discrete Output N-2

The I/O Assembly data attribute for the output data status bits has the format shown below.

Table 6-12.10 I/O Assembly Data Attribute Format – Output Data Status Bits

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
41	0	Reserved							Discrete Output Status1
42	0	Reserved							Discrete Output Status2
43	0	Reserved				Discrete Output Status4	Discrete Output Status3	Discrete Output Status2	Discrete Output Status1
44	0	Discrete Output Status8	Discrete Output Status7	Discrete Output Status6	Discrete Output Status5	Discrete Output Status4	Discrete Output Status3	Discrete Output Status2	Discrete Output Status1
45	0	Discrete Output Status8	Discrete Output Status7	Discrete Output Status6	Discrete Output Status5	Discrete Output Status4	Discrete Output Status3	Discrete Output Status2	Discrete Output Status1
	1	Discrete Output Status16	Discrete Output Status15	Discrete Output Status14	Discrete Output Status13	Discrete Output Status12	Discrete Output Status11	Discrete Output Status10	Discrete Output Status9

General Purpose Discrete I/O Device, Type: 07<sub>Hex</sub>

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
46	0	Discrete Output Status8	Discrete Output Status7	Discrete Output Status6	Discrete Output Status5	Discrete Output Status4	Discrete Output Status3	Discrete Output Status2	Discrete Output Status1
	1	Discrete Output Status16	Discrete Output Status15	Discrete Output Status14	Discrete Output Status13	Discrete Output Status12	Discrete Output Status11	Discrete Output Status10	Discrete Output Status9
	2	Discrete Output Status24	Discrete Output Status23	Discrete Output Status22	Discrete Output Status21	Discrete Output Status20	Discrete Output Status19	Discrete Output Status18	Discrete Output Status17
	3	Discrete Output Status32	Discrete Output Status31	Discrete Output Status30	Discrete Output Status29	Discrete Output Status28	Discrete Output Status27	Discrete Output Status26	Discrete Output Status25
47	0	Discrete Output Status8	Discrete Output Status7	Discrete Output Status6	Discrete Output Status5	Discrete Output Status4	Discrete Output Status3	Discrete Output Status2	Discrete Output Status1
	•								
	M	Reserved						Discrete Output Status N	Discrete Output Status N-1

The I/O Assembly data attribute for the input data with one input status bit and one output status bit has the format shown below.

**Table 6-12.11 I/O Assembly Data Attribute Format – Assy with 1 Input & Output Status Bit**

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
52	0	Discrete Input Status	Discrete Output Status	Reserved				Discrete Input2	Discrete Input1
53	0	Discrete Input Status	Discrete Output Status	Reserved		Discrete Input4	Discrete Input3	Discrete Input 2	Discrete Input1
54	0	Discrete Input8	Discrete Input7	Discrete Input6	Discrete Input5	Discrete Input4	Discrete Input3	Discrete Input2	Discrete Input1
	1	Discrete Input Status	Discrete Output Status	Reserved					
55	0	Discrete Input8	Discrete Input7	Discrete Input6	Discrete Input5	Discrete Input4	Discrete Input3	Discrete Input2	Discrete Input1
	1	Discrete Input16	Discrete Input15	Discrete Input14	Discrete Input13	Discrete Input12	Discrete Input11	Discrete Input10	Discrete Input9
	2	Discrete Input Status	Discrete Output Status	Reserved					

General Purpose Discrete I/O Device, Type: 07<sub>Hex</sub>

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
56	0	Discrete Input8	Discrete Input7	Discrete Input6	Discrete Input5	Discrete Input4	Discrete Input3	Discrete Input2	Discrete Input1
	1	Discrete Input16	Discrete Input15	Discrete Input14	Discrete Input13	Discrete Input12	Discrete Input11	Discrete Input10	Discrete Input9
	2	Discrete Input24	Discrete Input23	Discrete Input22	Discrete Input21	Discrete Input20	Discrete Input19	Discrete Input18	Discrete Input17
	3	Discrete Input32	Discrete Input31	Discrete Input30	Discrete Input29	Discrete Input28	Discrete Input27	Discrete Input26	Discrete Input25
	4	Discrete Input Status	Discrete Output Status	Reserved					
57	0	Discrete Input8	Discrete Input7	Discrete Input6	Discrete Input5	Discrete Input4	Discrete Input3	Discrete Input2	Discrete Input1
	•								
	M	Discrete Input Status	Discrete Output Status	Reserved			Discrete Input N	Discrete Input N-1	Discrete Input N-2

The I/O Assembly data attribute for the input data with multiple input status bits and multiple output status bits has the format shown below.

Table 6-12.12 I/O Assembly Data Attribute Format – Assy w/Multiple Input &amp; Output Status Bits

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
62	0	Reserved		Discrete Output Status2	Discrete Output Status1	Discrete Input Status2	Discrete Input Status1	Discrete Input2	Discrete Input1
63	0	Discrete Input Status4	Discrete Input Status3	Discrete Input Status2	Discrete Input Status1	Discrete Input4	Discrete Input3	Discrete Input2	Discrete Input1
	1	Reserved				Discrete Output Status4	Discrete Output Status3	Discrete Output Status2	Discrete Output Status1
64	0	Discrete Input8	Discrete Input7	Discrete Input6	Discrete Input5	Discrete Input4	Discrete Input3	Discrete Input2	Discrete Input1
	1	Discrete Input Status8	Discrete Input Status7	Discrete Input Status6	Discrete Input Status5	Discrete Input Status4	Discrete Input Status3	Discrete Input Status2	Discrete Input Status1
	2	Discrete Output Status8	Discrete Output Status7	Discrete Output Status6	Discrete Output Status5	Discrete Output Status4	Discrete Output Status3	Discrete Output Status2	Discrete Output Status1

General Purpose Discrete I/O Device, Type: 07<sub>Hex</sub>

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
65	0	Discrete Input8	Discrete Input7	Discrete Input6	Discrete Input5	Discrete Input4	Discrete Input3	Discrete Input2	Discrete Input1
	1	Discrete Input16	Discrete Input15	Discrete Input14	Discrete Input13	Discrete Input12	Discrete Input11	Discrete Input10	Discrete Input9
	2	Discrete Input Status8	Discrete Input Status7	Discrete Input Status6	Discrete Input Status5	Discrete Input Status4	Discrete Input Status3	Discrete Input Status2	Discrete Input Status1
	3	Discrete Input Status16	Discrete Input Status15	Discrete Input Status14	Discrete Input Status15	Discrete Input Status12	Discrete Input Status11	Discrete Input Status10	Discrete Input Status9
	4	Discrete Output Status8	Discrete Output Status7	Discrete Output Status6	Discrete Output Status5	Discrete Output Status4	Discrete Output Status3	Discrete Output Status2	Discrete Output Status1
	5	Discrete Output Status16	Discrete Output Status15	Discrete Output Status14	Discrete Output Status13	Discrete Output Status12	Discrete Output Status11	Discrete Output Status10	Discrete Output Status9

The I/O Assembly data attribute for the input data with single input status bit and multiple output status bits has the format shown below.

Table 6-12.13 I/O Assembly Data Attribute Format –Assy w/Mixed Input &amp; Output Status Bits

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
70	0	Discrete Input Status	Reserved					Discrete Output Status1	Discrete Input1
71	0	Discrete Input Status	Reserved					Discrete Output Status1	Discrete Input1
72	0	Discrete Input Status	Reserved			Discrete Output Status2	Discrete Output Status1	Discrete Input2	Discrete Input1
73	0	Discrete Input Status	Reserved	Discrete Output Status2	Discrete Output Status1	Discrete Input4	Discrete Input3	Discrete Input2	Discrete Input1
74	0	Discrete Output Status4	Discrete Output Status3	Discrete Output Status2	Discrete Output Status1	Discrete Input4	Discrete Input3	Discrete Input2	Discrete Input1
	1	Discrete Input Status	Reserved						
75	0	Discrete Input8	Discrete Input7	Discrete Input6	Discrete Input5	Discrete Input4	Discrete Input3	Discrete Input2	Discrete Input1
	1	Discrete Input Status	Reserved			Discrete Output Status4	Discrete Output Status3	Discrete Output Status2	Discrete Output Status1

General Purpose Discrete I/O Device, Type: 07<sub>Hex</sub>

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
76	0	Discrete Input8	Discrete Input7	Discrete Input6	Discrete Input5	Discrete Input4	Discrete Input3	Discrete Input2	Discrete Input1
	1	Discrete Output Status8	Discrete Output Status7	Discrete Output Status6	Discrete Output Status5	Discrete Output Status4	Discrete Output Status3	Discrete Output Status2	Discrete Output Status1
	2	Discrete Input Status	Reserved						
77	0	Discrete Input8	Discrete Input7	Discrete Input6	Discrete Input5	Discrete Input4	Discrete Input3	Discrete Input2	Discrete Input1
	1	Discrete Input16	Discrete Input15	Discrete Input14	Discrete Input13	Discrete Input12	Discrete Input11	Discrete Input10	Discrete Input9
	2	Discrete Output Status8	Discrete Output Status7	Discrete Output Status6	Discrete Output Status5	Discrete Output Status4	Discrete Output Status3	Discrete Output Status2	Discrete Output Status1
	3	Discrete Input Status	Reserved						
78	0	Discrete Input8	Discrete Input7	Discrete Input6	Discrete Input5	Discrete Input4	Discrete Input3	Discrete Input2	Discrete Input1
	1	Discrete Input16	Discrete Input15	Discrete Input14	Discrete Input13	Discrete Input12	Discrete Input11	Discrete Input10	Discrete Input9
	2	Discrete Output Status8	Discrete Output Status7	Discrete Output Status6	Discrete Output Status5	Discrete Output Status4	Discrete Output Status3	Discrete Output Status2	Discrete Output Status1
	3	Discrete Output Status16	Discrete Output Status15	Discrete Output Status14	Discrete Output Status13	Discrete Output Status12	Discrete Output Status11	Discrete Output Status10	Discrete Output Status9
	4	Discrete Input Status	Reserved						

## 6-12.6 Mapping I/O Assembly Data Attribute Components

The following table indicates the I/O assembly Data attribute mapping for the General Purpose Discrete I/O device for the input assemblies with a single status bit.

Table 6-12.14 I/O Assembly Data Mapping – Single Status Bit Assemblies

Data Component Name	Class		Instance Number	Attribute	
	Name	Number		Name	Number
Discrete InputN	Discrete Input Point	08 <sub>hex</sub>	N	Value	3
Status or Discrete Input Status	Discrete Input Group	1D <sub>hex</sub>	1	Status	5

**General Purpose Discrete I/O Device, Type: 07<sub>Hex</sub>**

**Important:** If I/O Assembly instances 7, 17, 27, 37, 47 or 57 are supported, the “Max Instance” attribute at the class level of the Discrete Input Point class or of the Discrete Output Point class must be supported.

The following table indicates the I/O assembly Data attribute mapping for the General Purpose Discrete I/O device for the input assemblies with multiple status bits.

**Table 6-12.15 I/O Assembly Data Mapping – Multiple Status Bit Assemblies**

Data Component Name	Class		Instance Number	Attribute	
	Name	Number		Name	Number
Discrete InputN	Discrete Input Point	08 <sub>hex</sub>	N	Value	3
StatusN or Discrete Input StatusN	Discrete Input Point	08 <sub>hex</sub>	N	Status	4

The following table indicates the I/O assembly Data attribute mapping for the General Purpose Discrete I/O device for the output assemblies.

**Table 6-12.16 I/O Assembly Data Mapping – Output Assemblies**

Data Component Name	Class		Instance Number	Attribute	
	Name	Number		Name	Number
Discrete OutputN	Discrete Output Point	09 <sub>hex</sub>	N	Value	3
Discrete Output StatusN	Discrete Output Point	09 <sub>hex</sub>	N	Status	4
Discrete Output Status	Discrete Output Group	1E <sub>hex</sub>	1	Status	5

## 6-12.7 Defining Device Configuration

Primary public interface to the Input Filter Selection parameter is accessed by the Discrete Output Group Object.

### 6-12.7.1 Input Configuration

There are no configuration parameters defined for the discrete inputs.

## 6-12.8 Output Configuration Assembly Instances

The following table identifies the output configuration assembly instance supported by the General Purpose Discrete I/O device.

**Table 6-12.17 Output Configuration Assembly Instances**

Number	Type	Name
40	Configuration	Output Configuration

## 6-12.9 Output Configuration Assembly Data Attribute Format

The Output Configuration Assembly Data attribute (typical throughout the document) has the format shown below.

**Table 6-12.18 Output Configuration Assembly Data Attribute Format**

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
40	0	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Idle Action	Fault Action

## 6-12.10 Mapping Output Configuration Assembly Data Attribute Components

The output configuration is accessed by instances of the Assembly Object Class. The following table indicates the output configuration assembly Data attribute mapping for the General Purpose Discrete I/O device.

**Table 6-12.19 Output Configuration Assembly Data Mapping**

Configuration Parameter Name	Class		Instance Number	Attribute	
	Name	Number		Name	Number
Fault Action	Discrete Output Group	1E <sub>hex</sub>	1	Fault Action	7
Idle Action	Discrete Output Group	1E <sub>hex</sub>	1	Idle Action	9

The following table shows the effect of the Fault State and Idle State parameters on behavior.

**Table 6-12.20 Effect of Fault State/Idle State on Behavior**

Parameter	Effect on behavior
Fault Action	Indicates whether the Fault Value or the last state is to be placed at the output in the event of a fault. All Discrete Outputs in the device are set to the same Fault Action. <b>Note:</b> Fault Value cannot be configured via this assembly. The default is 0.
Idle Action	Indicates whether the Idle Value or the last state is to be placed at the output in the event of an idle. All Discrete Outputs in the device are set to the same Idle Action. <b>Note:</b> Idle Value cannot be configured via this assembly. The default is 0.

**General Purpose Discrete I/O Device, Type: 07<sub>Hex</sub>**

The following portion of an example EDS shows the information necessary to define the output configuration parameters for the General Purpose Discrete I/O device.

## [ Params ]

```
Param1=$ Fault Action
0,                                $ reserved
6,"20 1E 24 01 30 07",           $ Link Path Size and Link Path
0x0000,                            $ No support for settable path,
                                    $ enumerated strings, scaling,
                                    $ scaling link, or real time
                                    $ update of value. Value is
                                    $ gettable and settable.
0xC1,                                $ Data Type
1,                                $ Data Size
"Fault Action",                  $ Parameter Name
"",                               $ Units String not used
"",                               $ Help string not used
0,1,0;                            $ Min, Max, and Default values
1,1,1,0,0,0,0,0,0;               $ Not used

Param2=                           $ Idle Action
0,                                $ reserved
6,"20 1E 24 01 30 09",           $ Link Path Size and Link Path
0x00,                            $ No support for settable path,
                                    $ enumerated strings, scaling,
                                    $ scaling link, or real time
                                    $ update of value. Value is
                                    $ gettable and settable.
4,                                $ Data Type
1,                                $ Data Size
"Idle Action",                  $ Parameter Name
"",                               $ Units String not used
"",                               $ Help string not used
0,1,0;                            $ Min, Max, and Default values
1,1,1,0,0,0,0,0,0;               $ Not used

[Groups]                         $ No need to support
```

## 6-13 Communications Adapter Device

### Device Type: 0C Hex

The Communications Adapter device type acts as a gateway from the CIP network to other technologies. Traditionally, a gateway connects to foreign networks (for example, RS-232) or backplanes (for example, VME). The technologies involved greatly affect the gateway modeling and definition. Initially, some devices will be defined as Communications Adapter devices, and the ODVA and CI forum may create a specific device profile for devices with similar functions.

### 6-13.1 Object Model

The Object Model in Figure 6-13.2 represents the minimum support in a Communications Adapter. The table below indicates:

- the object classes present in this device
- whether or not the class is required
- the number of instances present in each class

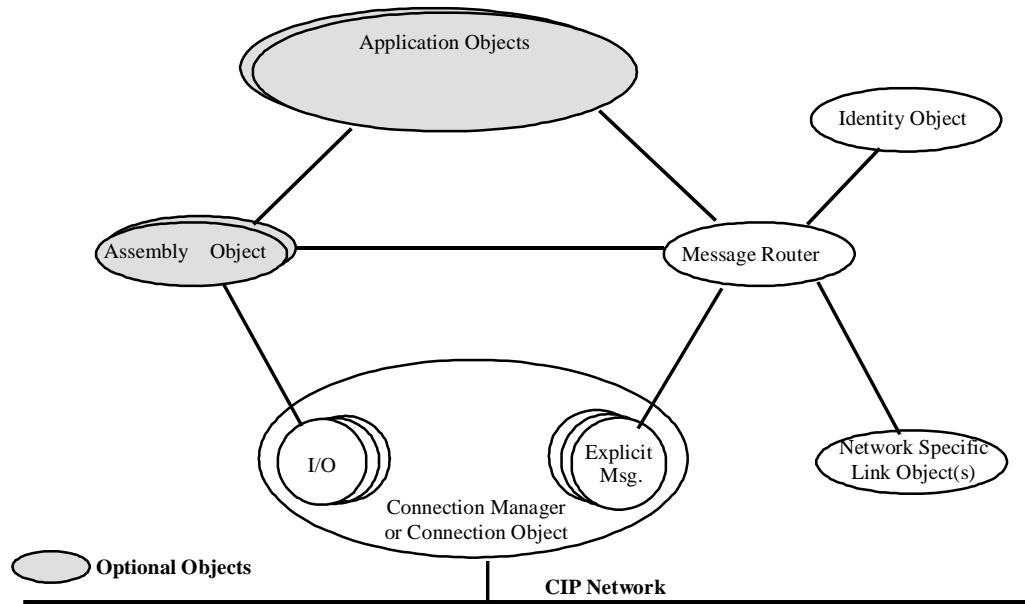
**Table 6-13.1 Objects Present in a Communications Adapter**

Object Class	Optional/Required	# of Instances
CIP Common Required Objects	Required	See Section 6-2.1
Assembly	Optional	Possibly 1 or more
Application	Optional	Possibly 1 or more

The Communications Adapter profile cannot specify the definition of the Assembly Object or the type of application objects necessary for device operation. This portion of the device profile must be supplied by the product developer as described in Sections 6-1 through 6-7 inclusive. Any Assembly instances created must be in the vendor-specific range (64<sub>hex</sub> - C7<sub>hex</sub>). Application objects may be public, vendor-specific, or both.

**Communications Adapter Device, Type: 0C<sub>Hex</sub>**

**Figure 6-13.2 Object Model for the Communications Adapter**



## **6-14 General Purpose Analog I/O Device**

The profile for this device type will be defined by the Open DeviceNet Vendor Association, Inc. and ControlNet International.

## **6-15 AC/DC Drive Devices**

**AC Drives Device Type: 02 Hex**

**DC Drives Device Type: 13 Hex**

These device profiles describe standard objects and behavior for AC and DC drives including Standard Scalar (V/Hz) AC, Vector AC, and DC Drives.

The functionality of drives covered includes:

- Open loop speed (frequency) control
- Closed loop speed (frequency) control
- Torque control
- No position control

These profiles make the drives inter-operable, but not directly interchangeable without doing drive configuration through the drive local interface, a network configuration tool or other means of configuration outside the CIP interface.

The AC and DC Drive profiles are part of a “Hierarchy of Motor Control Devices” that is supported by CIP. This hierarchy includes:

- Contactors and Across the Line Motor Starters
- Soft Starters
- AC and DC Drives
- Servo Drives

Devices within this hierarchy all use a common “Control Supervisor” object to control the state behavior of the device. All but the low level Contactors and Across the Line Motor Starters also use a common “Motor Data” object to store information about the motor to be controlled. The Hierarchy of Motor Control Devices also supports a hierarchy of IO Assembly Instance definitions. Assembly instances are numbered within the hierarchy so that each device type is assigned a range of Assembly Instance numbers, with higher functionality devices supporting higher instance numbers. Devices in the hierarchy can choose to support some IO Assembly Instance numbers that are lower than theirs in the hierarchy. For example an AC Drive may choose to support some IO Assemblies that are defined in the Starter Profile to make it easier to interchange drives and starters in a system.

### **6-15.1 Multiple axes on one drive**

It is possible to implement several axes of control on one physical drive unit. A separate MAC ID must be assigned to each axis so each axis is treated as separate CIP node.

## 6-15.2 Object Model

The Object Model in figure 6-15.2 represents an AC or DC Drive. The table below indicates

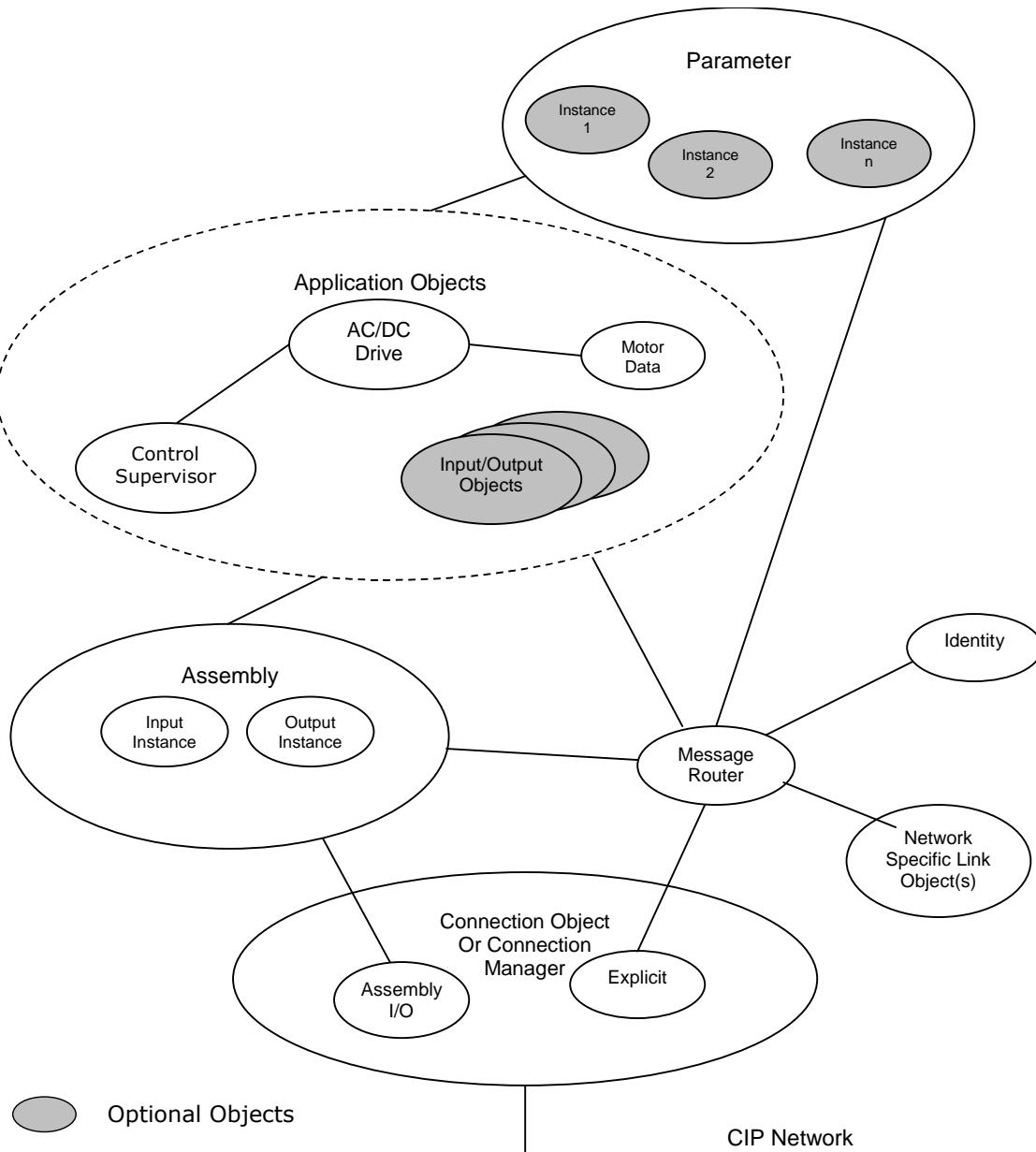
- the object classes present in this device
- whether or not the class is required
- the number of instances present in each class

Chapter 5, The CIP Object Library, provides more details about these objects.

**Table 6-15.1 Objects Present in an AC/DC Drive**

Object Class	Optional / Required	# of Instances
Message Router	Optional	1
CIP Common Required Objects	Required	See Section 6-2.1
Assembly	Required	2
Control Supervisor	Required	1
AC/DC Drive	Required	1
Motor Data	Required	1
Parameter	Optional	-
Parameter Group	Optional	-
Discrete Input	Optional	-
Discrete Output	Optional	-
Analog Input	Optional	-
Analog Output	Optional	-

**Figure 6-15.2 Object Model for AC and DC Drives**



### 6-15.3 How Objects Affect Behavior

The objects in this device affect the device's behavior as shown in the following table.

**Table 6-15.3 Object Effect on Behavior**

Object	Effect on Behavior
CIP Common Required	See Section 6-2.2 for details.
Assembly	Defines I/O data format
Control Supervisor	Manages drive functions, operational states and control
AC/DC Drive	Provides drive configuration
Motor Data	Defines motor data for the motor connected to this device
Parameter	Provides a public interface to device configuration data
Parameter Group	Provides an aid to device configuration
Discrete Input	Defines the behavior of discrete inputs on this device
Discrete Output	Defines the behavior of discrete outputs on this device
Analog Input	Defines the behavior of analog inputs on this device
Analog Output	Defines the behavior of analog outputs on this device

### 6-15.4 Defining Object Interfaces

The objects in this device have the interfaces listed in the following table.

**Table 6-15.4 Object Interfaces**

Object	Interface
CIP Common Required	See Section 6-2.3 for details.
Assembly	I/O Connection or Message Router
Control Supervisor	Message Router, Assembly or Parameter Object
AC/DC Drive	Message Router, Assembly or Parameter Object
Motor Data	Message Router or Parameter Object
Parameter	Message Router
Discrete Input	Message Router or Assembly
Discrete Output	Message Router or Assembly
Analog Input	Message Router or Assembly
Analog Output	Message Router or Assembly

## 6-15.5 I/O Assembly Instances

The IO Assembly Instance definitions in this section define the format of the “data” attribute (attribute 3) for IO Assembly Instances. Through the use of predefined instance definitions, IO Assemblies support a hierarchy of motor control devices. The device hierarchy includes motor starters, soft starters, AC and DC drives, and servo drives. Assembly Instances are numbered within the hierarchy so that each device type is assigned a range of Assembly Instance numbers, with higher functionality devices supporting higher instance numbers. **Devices in the hierarchy can choose to support instance numbers that are lower than theirs in the hierarchy.** For example an AC drive may choose to support some IO Assemblies that are defined in the starter profile to make it easier to interchange starters and drives within a system. The following table shows the Assembly Instance numbering for the motor control device hierarchy.

**Table 6-15.5 I/O Assembly Instance Numbering Ranges**

Profile	I/O Type	Instance Range	Instances within hierarchy that may be implemented for this product type.
AC Motor Starter	Output	1-19	1-19
Soft Start Starter	Input	50-69	50-69
AC or DC Drive	Output	20-29	1-29
	Input	70-79	50-79
Servo Drive	Output	30-49	1-49
	Input	80-99	50-99

The following IO Assembly Instances are defined for AC and DC Drives.

**Table 6-15.6 I/O Assembly Instances**

Number		Required/Optional	Type	Name
decimal	hex			
20	14	Required	Output	Basic Speed Control Output
21	15	Optional	Output	Extended Speed Control Output
22	16	Optional	Output	Speed and Torque Control Output
23	17	Optional	Output	Extended Speed and Torque Control Output
24	18	Optional	Output	Process Control Output
25	19	Optional	Output	Extended Process Control Output
70	46	Required	Input	Basic Speed Control Input
71	47	Optional	Input	Extended Speed Control Input
72	48	Optional	Input	Speed and Torque Control Input
73	49	Optional	Input	Extended Speed and Torque Control Input
74	4A	Optional	Input	Process Control Input
75	4B	Optional	Input	Extended Process Control Input

If a bit is not used in an IO Assembly, it is reserved for use in other Assemblies. Reserved bits in Output Assemblies are ignored by the consuming device. Reserved bits in Input Assemblies are set to zero by the producing device.

Reserved bits in the IO Assembly Data Attribute Format Tables are shaded.

### 6-15.5.1 Connection Paths to I/O Assembly Instances

The IO Assembly Instances are chosen for IO Connections by setting the “produced\_connection\_path” (attribute 14) and “consumed\_connection\_path” (attribute 16) attributes in the appropriate connection object.

AC and DC Drives use the Symbolic Segment Type (see Appendix C) to specify paths to the IO Assembly Instances in the Motor Control Hierarchy. IO Assembly Instances are represented by ASCII strings that contain the hex number of the Assembly Instance whose path is to be chosen.

The following example shows the Symbolic Segment used to specify Output Assembly Instance 20 (14 hex).

**Figure 6-15.7 Symbolic Segment Example for Output Assy**

Segment Type		First Character					Second Character				
0	1	1	0	0	0	1	0	0	0	0	1
Segment type type (symbolic)	Symbol size in bytes (2 bytes)	ASCII 1 (31 hex)					ASCII 4 (34 hex)				

### 6-15.6 I/O Assembly Data Attribute Format

The I/O Assembly Data Attributes have the format shown below.

**Table 6-15.8 I/O Assembly Data Attribute Format**

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
20	0						Fault		Run
	1						Reset		Fwd
	2						Speed Reference (Low Byte)		
	3						Speed Reference (High Byte)		
21	0		NetRef	NetCtrl			Fault	Run	Run
	1						Reset	Rev	Fwd
	2						Speed Reference (Low Byte)		
	3						Speed Reference (High Byte)		
22	0						Fault		Run
	1						Reset		Fwd
	2						Speed Reference (Low Byte)		
	3						Speed Reference (High Byte)		
	4						Torque Reference (Low Byte)		
	5						Torque Reference (High Byte)		

**AC/DC Drive Devices, Types: 02<sub>Hex</sub> and 13<sub>Hex</sub>**

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
23	0		NetRef	NetCtrl			Fault Reset	Run Rev	Run Fwd
	1								
	2								Speed Reference (Low Byte)
	3								Speed Reference (High Byte)
	4								Torque Reference (Low Byte)
	5								Torque Reference (High Byte)
24	0						Fault Reset		Run Fwd
	1								
	2								Speed Reference (Low Byte)
	3								Speed Reference (High Byte)
	4								Process Reference (Low Byte)
	5								Process Reference (High Byte)
25	0	NetProc	NetRef	NetCtrl			Fault Reset	Run Rev	Run Fwd
	1								Mode
	2								Speed Reference (Low Byte)
	3								Speed Reference (High Byte)
	4								Process Reference (Low Byte)
	5								Process Reference (High Byte)
70	0						Running1		Faulted
	1								
	2								Speed Actual (Low Byte)
	3								Speed Actual (High Byte)
71	0	At Reference	Ref From Net	Ctrl From Net	Ready	Running2 (Rev)	Running1 (Fwd)	Warning	Faulted
	1								Drive State
	2								Speed Actual (Low Byte)
	3								Speed Actual (High Byte)
72	0						Running1		Faulted
	1								
	2								Speed Actual (Low Byte)
	3								Speed Actual (High Byte)
	4								Torque Actual (Low Byte)
	5								Torque Actual (High Byte)
73	0	At Reference	Ref From Net	Ctrl From Net	Ready	Running2 (Rev)	Running1 (Fwd)	Warning	Faulted
	1								Drive State
	2								Speed Actual (Low Byte)
	3								Speed Actual (High Byte)
	4								Torque Actual (Low Byte)
	5								Torque Actual (High Byte)

AC/DC Drive Devices, Types: 02<sub>Hex</sub> and 13<sub>Hex</sub>

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
74	0						Running1		Faulted
	1								
	2					Speed Actual (Low Byte)			
	3					Speed Actual (High Byte)			
	4				Process Actual (Low Byte)				
	5				Process Actual (High Byte)				
75	0	At Reference	Ref From Net	Ctrl From Net	Ready	Running2 (Rev)	Running1 (Fwd)	Warning	Faulted
	1					Drive State			
	2					Speed Actual (Low Byte)			
	3					Speed Actual (High Byte)			
	4					Process Actual (Low Byte)			
	5					Process Actual (High Byte)			

**6-15.7 Mapping I/O Assembly Data Attribute Components**

The following table indicates the I/O Assembly Data Attribute mapping for AC and DC Drive Output Assemblies.

**Table 6-15.9 I/O Assembly Data Mapping – Output Assemblies**

Data Component Name	Class		Instance Number	Attribute	
	Name	Number		Name	Number
RunFwd	Control Supervisor	29 <sub>hex</sub>	1	Run1	3
RunRev	Control Supervisor	29 <sub>hex</sub>	1	Run2	4
Fault Reset	Control Supervisor	29 <sub>hex</sub>	1	FaultRst	12
NetCtrl	Control Supervisor	29 <sub>hex</sub>	1	NetCtrl	5
NetRef	AC/DC Drive	2A <sub>hex</sub>	1	NetRef	4
Net Proc	AC/DC Drive	2A <sub>hex</sub>	1	NetProc	5
Drive Mode	AC/DC Drive	2A <sub>hex</sub>	1	DriveMode	6
Speed Reference	AC/DC Drive	2A <sub>hex</sub>	1	SpeedRef	8
Torque Reference	AC/DC Drive	2A <sub>hex</sub>	1	TorqueRef	12
Process Reference	AC/DC Drive	2A <sub>hex</sub>	1	ProcessRef	14

The following table indicates the I/O Assembly Data Attribute mapping for AC and DC Drive Input Assemblies.

**Table 6-15.10 I/O Assembly Data Mapping – Input Assemblies**

Data Component Name	Class		Instance Number	Attribute	
	Name	Number		Name	Number
Faulted	Control Supervisor	29 <sub>hex</sub>	1	Faulted	10
Warning	Control Supervisor	29 <sub>hex</sub>	1	Warning	11
Running1 (Fwd)	Control Supervisor	29 <sub>hex</sub>	1	Running1	7
Running2 (Rev)	Control Supervisor	29 <sub>hex</sub>	1	Running2	8
Ready	Control Supervisor	29 <sub>hex</sub>	1	Ready	9
CtrlFromNet	Control Supervisor	29 <sub>hex</sub>	1	CtrlFromNet	15
Drive State	Control Supervisor	29 <sub>hex</sub>	1	State	6
Ref From Net	AC/DC Drive	2A <sub>hex</sub>	1	RefFromNet	29
At Reference	AC/DC Drive	2A <sub>hex</sub>	1	AtReference	3
Speed Actual	AC/DC Drive	2A <sub>hex</sub>	1	SpeedActual	7
Torque Actual	AC/DC Drive	2A <sub>hex</sub>	1	TorqueActual	11
Process Actual	AC/DC Drive	2A <sub>hex</sub>	1	ProcessActual	13

## 6-15.8 Defining Device Configuration

Public access to the Control Supervisor Object, the Motor Data Object, and the AC/DC Drive Object must be supported for configuration of an AC or DC drive. If supported, the optional Parameter Objects may be used to access the various configuration attributes in the Control Supervisor Object, the Motor Data Object, and the AC/DC Drive Object.

AC and DC drives may contain (but are not limited to) any of the Parameter Object instances listed in the table below. Suggested parameter names are also given in the table. The set of parameters instances that are supported by a drive should be numbered sequentially with lower instance numbers assigned to parameters that appear earlier in the table. Vendor specific parameter instances should be numbered sequentially following the instances that appear in the following table.

Parameter Object instances may be implemented as EDS file definitions, parameter stubs, or full parameter objects. See Chapter 5 of the CIP Common specification for a definition of the Parameter Object and an explanation of how it is used for configuration.

### 6-15.8.1 Mapping Parameter Object Data

The following table indicates the Parameter Object data mapping for an AC or DC Drive device.

**Table 6-15.11 Parameter Object Data Mapping**

Configuration Parameter Name	Class		Instance Number	Attribute	
	Name	Number		Name	Number
Motor Type	Motor Data	28 <sub>hex</sub>	1	MotorType	3
Motor Cat Number	Motor Data	28 <sub>hex</sub>	1	CatNumber	4
Motor Vendor	Motor Data	28 <sub>hex</sub>	1	Manufacturer	5
Motor Rated Cur	Motor Data	28 <sub>hex</sub>	1	RatedCurrent	6
Motor Rated Volt	Motor Data	28 <sub>hex</sub>	1	RatedVoltage	7
Motor Rated Pwr	Motor Data	28 <sub>hex</sub>	1	RatedPower	8
Motor Rated Freq	Motor Data	28 <sub>hex</sub>	1	RatedFreq	9
Motor Rated Temp	Motor Data	28 <sub>hex</sub>	1	RatedTemp	10
Motor Max Speed	Motor Data	28 <sub>hex</sub>	1	MaxSpeed	11
Motor Pole Count	Motor Data	28 <sub>hex</sub>	1	PoleCount	12
Motor Torq Const	Motor Data	28 <sub>hex</sub>	1	TorqConstant	13
Motor Inertia	Motor Data	28 <sub>hex</sub>	1	Inertia	14
Motor Base Speed	Motor Data	28 <sub>hex</sub>	1	BaseSpeed	15
Motor Field Cur	Motor Data	28 <sub>hex</sub>	1	RatedFieldCur	16
Min Field Cur	Motor Data	28 <sub>hex</sub>	1	MinFieldCur	17
Rated Field Volt	Motor Data	28 <sub>hex</sub>	1	RatedFieldVolt	18
Service Factor	Motor Data	28 <sub>hex</sub>	1	ServiceFactor	19
Network Control	Control Supervisor	29 <sub>hex</sub>	1	NetCtrl	5
Drive State	Control Supervisor	29 <sub>hex</sub>	1	State	6
Running Fwd	Control Supervisor	29 <sub>hex</sub>	1	Running1	7
Running Rev	Control Supervisor	29 <sub>hex</sub>	1	Running2	8
Ready	Control Supervisor	29 <sub>hex</sub>	1	Ready	9
Faulted	Control Supervisor	29 <sub>hex</sub>	1	Faulted	10
Warning	Control Supervisor	29 <sub>hex</sub>	1	Warning	11
Fault Reset	Control Supervisor	29 <sub>hex</sub>	1	FaultRst	12
Fault Code	Control Supervisor	29 <sub>hex</sub>	1	FaultCode	13
Warning Code	Control Supervisor	29 <sub>hex</sub>	1	WarningCode	14
Control From Net	Control Supervisor	29 <sub>hex</sub>	1	CtrlFromNet	15
DN Fault Mode	Control Supervisor	29 <sub>hex</sub>	1	DNFaultMode	16
Force Fault	Control Supervisor	29 <sub>hex</sub>	1	ForceFault/Trip	17
Force Status	Control Supervisor	29 <sub>hex</sub>	1	ForceStatus	18
At Reference	AC/DC Drive	2A <sub>hex</sub>	1	AtReference	3
Network Ref	AC/DC Drive	2A <sub>hex</sub>	1	NetRef	4
Network Process	AC/DC Drive	2A <sub>hex</sub>	1	NetProc	5
Drive Mode	AC/DC Drive	2A <sub>hex</sub>	1	DriveMode	6
Speed Actual	AC/DC Drive	2A <sub>hex</sub>	1	SpeedActual	7
Speed Reference	AC/DC Drive	2A <sub>hex</sub>	1	SpeedRef	8
Current Actual	AC/DC Drive	2A <sub>hex</sub>	1	CurrentActual	9
Current Limit	AC/DC Drive	2A <sub>hex</sub>	1	CurrentLimit	10
Torque Actual	AC/DC Drive	2A <sub>hex</sub>	1	TorqueActual	11
Torque Reference	AC/DC Drive	2A <sub>hex</sub>	1	TorqueRef	12
Process Actual	AC/DC Drive	2A <sub>hex</sub>	1	ProcessActual	13

AC/DC Drive Devices, Types: 02<sub>Hex</sub> and 13<sub>Hex</sub>

Configuration Parameter	Class		Instance Number	Attribute	
	Name	Name		Number	Number
Process Reference	AC/DC Drive	2A <sub>hex</sub>	1	ProcessRef	14
Power Actual	AC/DC Drive	2A <sub>hex</sub>	1	PowerActual	15
Input Voltage	AC/DC Drive	2A <sub>hex</sub>	1	InputVoltage	16
Output Voltage	AC/DC Drive	2A <sub>hex</sub>	1	OutputVoltage	17
Accel Time	AC/DC Drive	2A <sub>hex</sub>	1	AccelTime	18
Decel Time	AC/DC Drive	2A <sub>hex</sub>	1	DecelTime	19
Low Speed Limit	AC/DC Drive	2A <sub>hex</sub>	1	LowSpdLimit	20
High Speed Limit	AC/DC Drive	2A <sub>hex</sub>	1	HighSpdLimit	21
Speed Scale	AC/DC Drive	2A <sub>hex</sub>	1	SpeedScale	22
Current Scale	AC/DC Drive	2A <sub>hex</sub>	1	CurrentScale	23
Torque Scale	AC/DC Drive	2A <sub>hex</sub>	1	TorqueScale	24
Process Scale	AC/DC Drive	2A <sub>hex</sub>	1	ProcessScale	25
Power Scale	AC/DC Drive	2A <sub>hex</sub>	1	PowerScale	26
Voltage Scale	AC/DC Drive	2A <sub>hex</sub>	1	VoltageScale	27
Time Scale	AC/DC Drive	2A <sub>hex</sub>	1	TimeScale	28
Ref From Net	AC/DC Drive	2A <sub>hex</sub>	1	RefFromNet	29
Proc From Net	AC/DC Drive	2A <sub>hex</sub>	1	ProcFromNet	30
Field I or V	AC/DC Drive	2A <sub>hex</sub>	1	FieldIorV	31
Field Voltage Ratio	AC/DC Drive	2A <sub>hex</sub>	1	FieldVoltRatio	32
Field Cur Set Pt	AC/DC Drive	2A <sub>hex</sub>	1	FieldCurSetPt	33
Field Weak Ena	AC/DC Drive	2A <sub>hex</sub>	1	FieldWkEnable	34
Field Cur Actual	AC/DC Drive	2A <sub>hex</sub>	1	FieldCurActual	35
Field Min Cur	AC/DC Drive	2A <sub>hex</sub>	1	FieldMinCur	36

**6-15.8.2 Parameter Group Objects**

AC and DC drives may contain (but are not limited to) any of the Parameter Group Object Instances listed in the table below. If Parameter Groups are supported, Parameter Instances should be grouped according to the object that their data is mapped to. For example, all Parameters Instances whose data maps to the Motor Data Object should be contained in the Motor Group (Parameter Group Object Instance 1).

Parameter Group Object instances may be implemented from an EDS file, or as actual Parameter Group objects from the device. See Chapter 5 of the CIP Common specification for a definition of the Parameter Group Object.

**Table 6-15.12 Parameter Group Object Instances**

<b>Parameter Group Name</b>	<b>Instance Number</b>	<b>Parameters in Group</b>
Motor	1	Motor Type Motor Cat Number Motor Vendor Motor Rated Cur Motor Rated Volt Motor Rated Pwr Motor Rated Freq Motor Rated Temp Motor Max Speed Motor Pole Count Motor Torq Const Motor Inertia Motor Base Speed Motor Field Cur Min Field Cur Rated Field Volt Rated Field Volt Service Factor
Supervisor	2	Network Control Drive State Running Fwd Running Rev Ready Faulted Warning Fault Reset Fault Code Warning Code Control From Net DN Fault Mode Force Fault Force Status

Parameter Group Name	Instance Number	Parameters in Group
Drive	3	At Reference Network Ref Network Process Drive Mode Speed Actual Speed Reference Current Actual Current Limit Torque Actual Torque Reference Process Actual Process Reference Power Actual Input Voltage Output Voltage Accel Time Decel Time Low Speed Limit High Speed Limit Speed Scale Current Scale Torque Scale Process Scale Power Scale Voltage Scale Time Scale Ref From Net Proc From Net Field I or V Field Voltage Ratio Field Cur Set Pt Field Weak Ena Field Cur Actual Field Min Cur

## **6-16      Servo Drive Device**

This device profile for a Servo Drive is presently under development. It is intended that this profile will be defined by the Servo Drive SIG of the Open DeviceNet Vendor Association, Inc. and ControlNet International.

## **6-17      Barcode Scanner Device**

The profile for this device type will be defined by the Open DeviceNet Vendor Association, Inc. and ControlNet International.

## 6-18 Position Controller Device

### Device Type: 10<sub>hex</sub>

A Position Controller controls the motion and position of a motor or linear actuator (servo, stepper, etc.). The position controller may or may not include an integrated drive. Positioning is achieved with a controlled motion profile. The motion profile is defined by Acceleration, Velocity, Deceleration, Position or Torque.

### 6-18.1 Object Model

The Object Model in figure 6-18.2 represents a Position Controller Device. The table below indicates:

- the object classes present in this device
- whether or not the class is required
- the number of instances present in each class

Chapter 5, Object Library, provides more details about these objects.

**Table 6-18.1 Objects Present in a Position Controller Device**

Object Class	Optional / Required	# of Instances
CIP Common Required Objects	Required	See Section 6-2.1
Position Controller Supervisor	Required	1 per Axis
Position Controller	Required	1 per Axis
Command Block	Optional	-
Block Sequencer	Optional	1 per Axis
Drive	Optional	1 per Axis
Motor Data	Optional	1 per Axis
Parameter	Optional	-

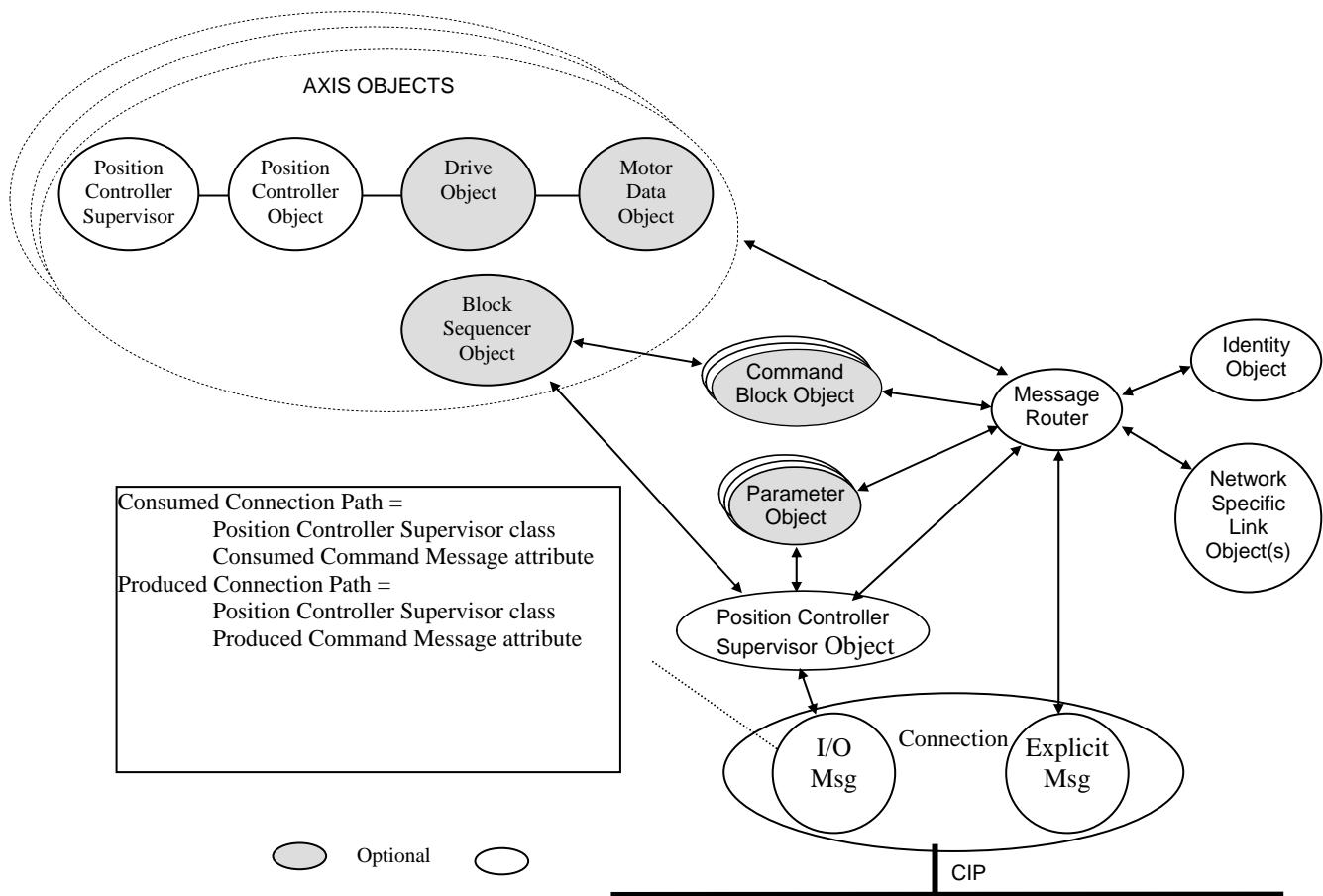
#### 6-18.1.1 Model Description

The object model shown below describes how the Position Controller device is controlled through CIP. Attributes can be set and queried in the normal manner for configuration. The Position Controller object handles the interface to the internal or external drive unit. The motor and drive units can be servo, stepper or some other method with optional feedback (open or closed loop).

In addition, the Command Block objects and the Block Sequencer object can be used to perform complex moves, modify attributes or wait for attributes to become valid. Command Blocks can be linked together to form a command chain with branching and looping supported. The user can download command block sequences during configuration and execute them at any time to perform complex moves or modify attributes. The Block Sequencer object is accessible through the I/O command message giving the user the ability to perform complex motion sequences from a PLC or scanner card.

Position Controller Device, Type: 10<sub>Hex</sub>

Figure 6-18.2 Object Model for a Position Controller



## 6-18.2 How Objects Affect Behavior

The objects for this device affect the device's behavior as shown in the table below.

Table 6-18.3 Object Effect on Behavior

Object	Effect on behavior
CIP Common Required	See Section 6-2.2 for details.
Position Controller Supervisor	Handles faults, home and registration inputs and modifies meaning of I/O data
Position Controller	Provides positioning control and manages interface to power amplifier
Block Sequencer	Executes command block sequences
Command Block	Defines the behavior of command blocks
Drive	Manages power amplifier
Motor Data	Configures the power amplifier for motor parameters
Parameter	Defines the behavior of Parameters

### 6-18.3 Defining Object Interfaces

The objects in this device have the interfaces listed in the following table:

**Table 6-18.4 Object Interfaces**

Object	Interface
CIP Common Required	See Section 6-2.3 for details.
Position Controller Supervisor	Message Router or Position Controller Supervisor Class
Position Controller	Message Router or Position Controller Supervisor Class
Block Sequencer	Message Router or Position Controller Supervisor Class
Command Block	Message Router or Position Controller Supervisor Class
Drive	Message Router or Position Controller Supervisor Class
Motor Data	Message Router or Position Controller Supervisor Class
Parameter	Message Router

### 6-18.4 I/O Connection Messages

The Position Controller Profile supports both command and response messages via the I/O connection. The produced and consumed paths specify the Position Controller Supervisor Class attributes as shown in Figure 6-18.2.

#### 6-18.4.1 Message Formats

The Position Controller Profile supports multiple axes per CIP node by allowing up to seven instances of each of the axis objects, in one Position Controller device. The axis objects are the 1) position controller supervisor, 2) position controller; 3) drive; 4) motor data; and 5) block sequencer. The I/O message can contain data from more than one axis object. The Command Axis Number and the Response Axis Number shown in the Message Format specifies the instance number of the axis object whose data is contained in the I/O message.

**Table 6-18.5 Command Message Format**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0					
0	Enable	Reg Arm	Hard Stop	Smooth Stop	Direction (V. Mode)	Incremental	Start Block	Load Data/ Start Profile					
1	Command Data 1												
2	Command Axis Number			Command Message Type									
3	Command Data 2												
4	Command Data 3												
5	Command Data 4												
6	Command Data 5												
7	Command Data 6												

**Position Controller Device, Type: 10<sub>Hex</sub>****Table 6-18.6 Response Message Format**

<b>Byte</b>	<b>Bit 7</b>	<b>Bit 6</b>	<b>Bit 5</b>	<b>Bit 4</b>	<b>Bit 3</b>	<b>Bit 2</b>	<b>Bit 1</b>	<b>Bit 0</b>
0	Enable	Reg Level	Home Level	Current Direction	General Fault	On Target Position	Block in execution	Profile in Progress
1						Response Data 1		
2	Load Complete	Block Fault	FE Fault	Negative Limit	Positive Limit	Rev Limit	Fwd Limit	Fault Input Fault
3			Response Axis Number			Response Message Type		
4						Response Data 2		
5						Response Data 3		
6						Response Data 4		
7						Response Data 5		

Note that a response message may contain data for a different axis number from what was contained in the command message. If an error is detected in the command or its requested response message, the response message shall be Command/Response Error Message Type 14 hex, not the requested response message.

#### **6-18.4.2 Definition of a Profile Move**

A profile move is a move that uses Acceleration, Target Velocity, and Deceleration to run at a Target Velocity or to a Target Position. In addition, the position controller device can output a Torque command. Whether or not the position controller device runs at a Target Velocity, to a Target Position or outputs a Torque command depends on the Operating Mode (Position Controller Object Attribute 3), to which the position controller device is set. The position controller device is set to Position, Velocity or Torque Mode using Position Controller Object Attribute 3.

#### **6-18.4.3 Starting a Profile Move**

The Position Controller Profile is mode-sensitive. The Position Controller Object Attribute 3 sets the mode of the controller to the following:

0 = Position (default)

1 = Velocity

2 = Torque

A profile move starts when the command message type for the specified mode is loaded and the Load Data/Start Profile bit transitions from zero to one. The table below shows the command message type, which starts a profile move for each mode.

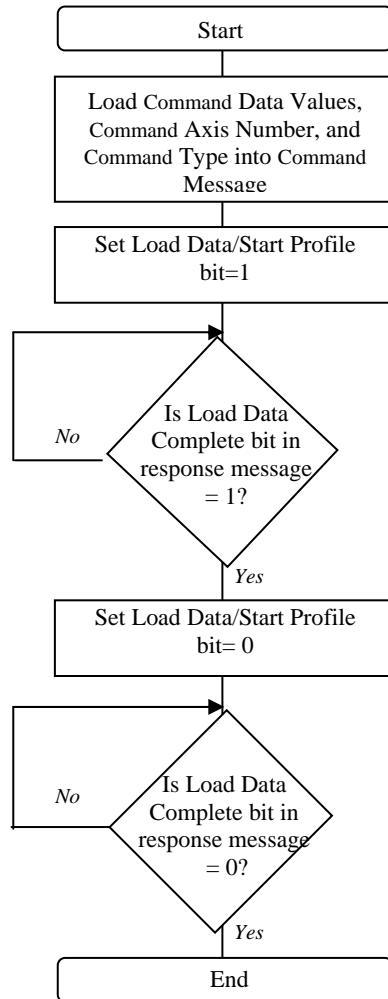
**Table 6-18.7 Command Message Type**

<b>Mode (Attribute 3)</b>	<b>Command Message Type which Starts Motion</b>
0 = Position	01 = Position
1 = Velocity	02 = Velocity
2 = Torque	05 = Torque

#### 6-18.4.4 I/O Handshaking Procedure

Proper handshaking between the client and the server is essential in I/O messaging to ensure that data sent to the position controller device is properly received. Two bits are used to provide handshaking between command and response messages. The recommended handshaking procedure for the client is described in the flowcharts in figure 6-18.8 and figure 6-18.9 below. The behavior required from the server to implement the handshake procedure is described in figure 6-18.10 and figure 6-18.11. Refer to the timing diagram in figure 6-18-12 below for representative timing of these bits during the handshake sequence

**Figure 6-18.8 Client Data Loading Procedure**



**Table 6-18.9 Client Profile Move Procedure**

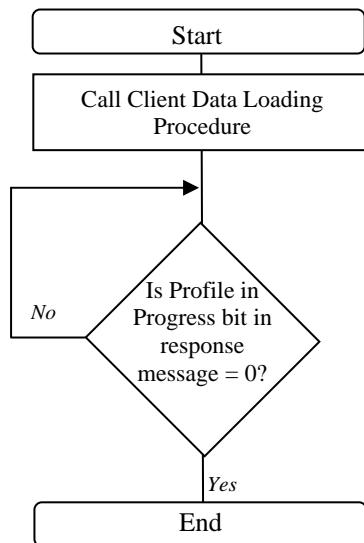
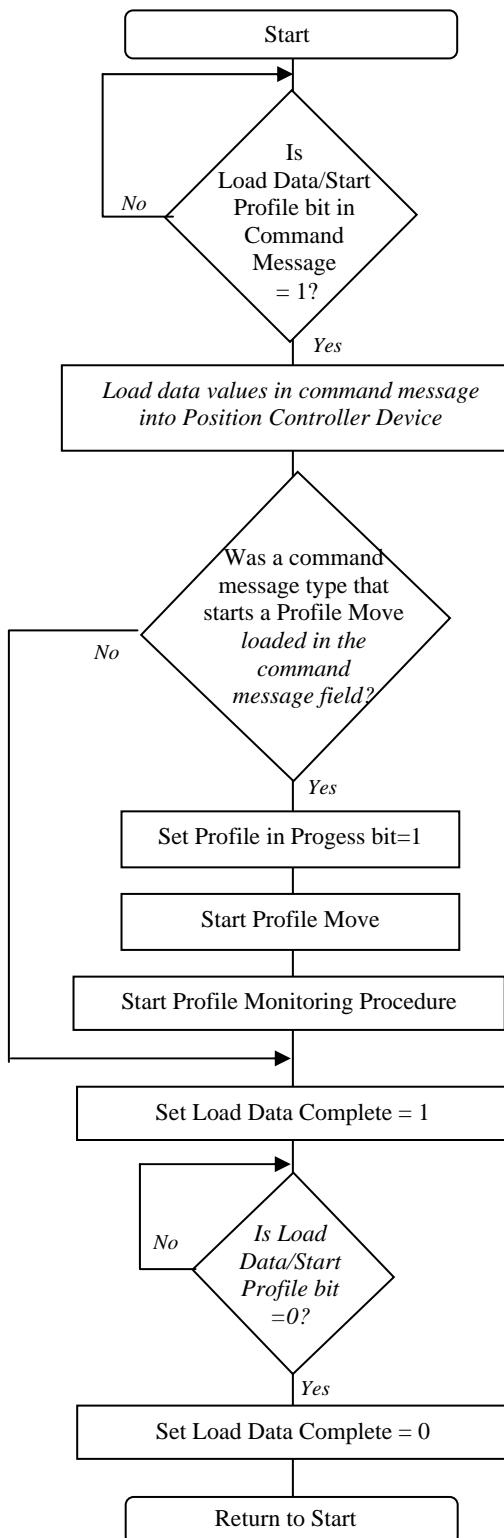
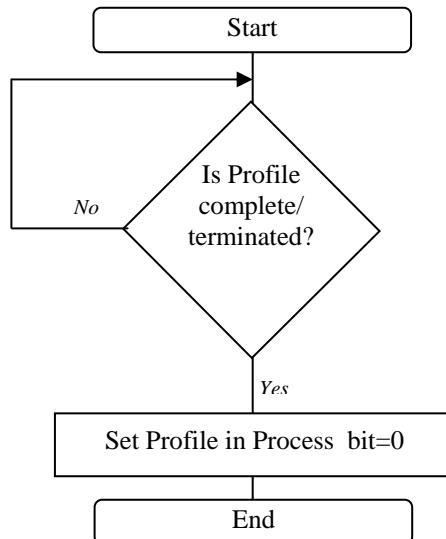
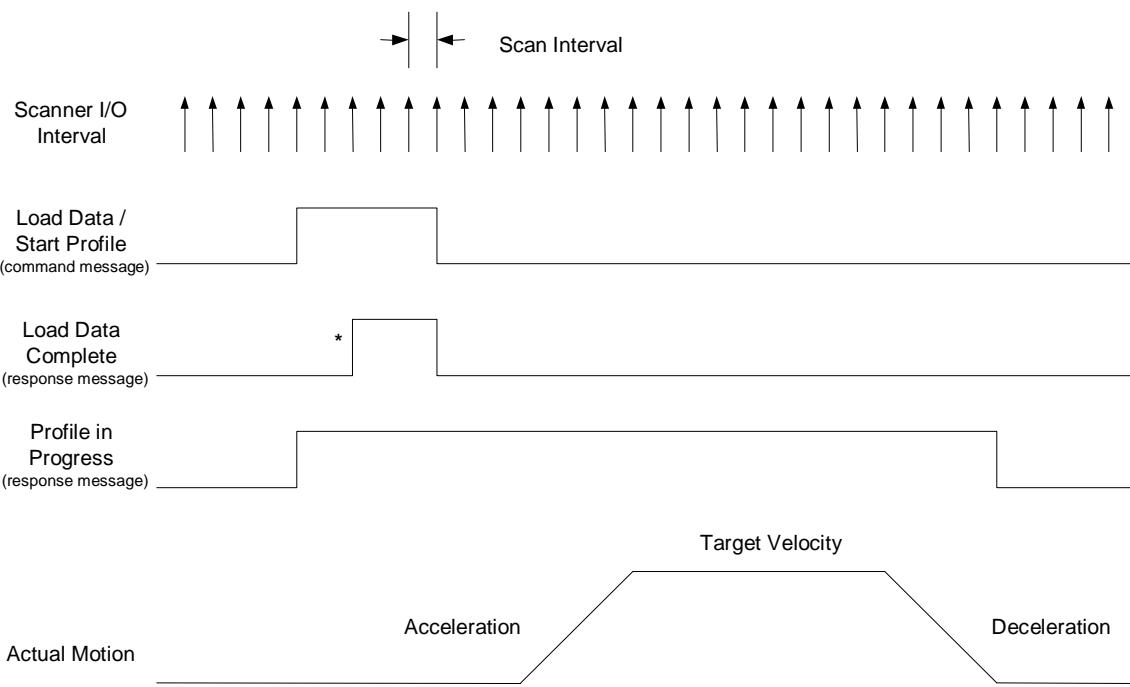


Table 6-18.10 Server Behavior



**Table 6-18.11 Profile Monitoring Procedure****Table 6-18.12 Timing Diagram**

\* Data Successfully Loaded into Position Control Object

## 6-18.5 I/O Connection Message Types

### 6-18.5.1 Command Message Types

The command message type is defined by byte 02 of the message. Byte 00 is the same for all command message types. Bytes 01 and 03 through 07 are defined by the Command Message Type code in byte 02. In message types 01 through 05, byte 03 defines the requested Response axis number and Response Message Type format. For message types 19 hex through 1F hex, the requested Response axis number and Response Message Type is the same as the Command axis number and Command Message Type.

**Table 6-18.13 Command Message Type\_01\_hex Target Position - Optional**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Enable	Reg Arm	Hard Stop	Smooth Stop	Direction (V. Mode)	Incremental	Start Block	Load Data/ Start Profile
1					Block #			
2	Command Axis Number				Command Message Type			
3	Response Axis Number				Response Message Type			
4					Target Position Low Byte			
5					Target Position Low Middle Byte			
6					Target Position High Middle Byte			
7					Target Position High Byte			

**Table 6-18.14 Command Message Type 02 hex Target Velocity - Optional**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Enable	Reg Arm	Hard Stop	Smooth Stop	Direction (V. Mode)	Incremental	Start Block	Load Data/ Start Profile
1					Block #			
2	Command Axis Number				Command Message Type			
3	Response Axis Number				Response Message Type			
4					Target Velocity Low Byte			
5					Target Velocity Low Middle Byte			
6					Target Velocity High Middle Byte			
7					Target Velocity High Byte			

**Table 6-18.15 Command Message Type 03 hex Acceleration - Optional**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Enable	Reg Arm	Hard Stop	Smooth Stop	Direction (V. Mode)	Incremental	Start Block	Load Data/ Start Profile
1					Block #			
2	Command Axis Number				Command Message Type			
3	Response Axis Number				Response Message Type			
4					Acceleration Low Byte			
5					Acceleration Low Middle Byte			
6					Acceleration High Middle Byte			
7					Acceleration High Byte			

**Position Controller Device, Type: 10<sub>Hex</sub>****Table 6-18.16 Command Message Type 04 hex Deceleration - Optional**

<b>Byte</b>	<b>Bit 7</b>	<b>Bit 6</b>	<b>Bit 5</b>	<b>Bit 4</b>	<b>Bit 3</b>	<b>Bit 2</b>	<b>Bit 1</b>	<b>Bit 0</b>
0	Enable	Reg Arm	Hard Stop	Smooth Stop	Direction (V. Mode)	Incremental	Start Block	Load Data/ Start Profile
1					Block #			
2	Command Axis Number				Command Message Type			
3	Response Axis Number				Response Message Type			
4					Deceleration Low Byte			
5					Deceleration Low Middle Byte			
6					Deceleration High Middle Byte			
7					Deceleration High Byte			

**Table 6-18.17 Command Message Type 05 hex Torque - Optional**

<b>Byte</b>	<b>Bit 7</b>	<b>Bit 6</b>	<b>Bit 5</b>	<b>Bit 4</b>	<b>Bit 3</b>	<b>Bit 2</b>	<b>Bit 1</b>	<b>Bit 0</b>
0	Enable	Reg Arm	Hard Stop	Smooth Stop	Direction (V. Mode)	Incremental	Start Block	Load Data/ Start Profile
1					Block #			
2	Command Axis Number				Command Message Type			
3	Response Axis Number				Response Message Type			
4					Torque Low Byte			
5					Torque Low Middle Byte			
6					Torque High Middle Byte			
7					Torque High Byte			

**Table 6-18.18 Command Message Type 19 hex Motor Data Attribute - Optional**

<b>Byte</b>	<b>Bit 7</b>	<b>Bit 6</b>	<b>Bit 5</b>	<b>Bit 4</b>	<b>Bit 3</b>	<b>Bit 2</b>	<b>Bit 1</b>	<b>Bit 0</b>
0	Enable	Reg Arm	Hard Stop	Smooth Stop	Direction (V. Mode)	Incremental	Start Block	Load Data/ Start Profile
1					Motor Data Attribute to Get			
2	Command Axis Number				Command Message Type			
3					Motor Data Attribute to Set			
4					Motor Data Attribute Value Low Byte			
5					Motor Data Attribute Value Low Middle Byte			
6					Motor Data Attribute Value High Middle Byte			
7					Motor Data Attribute Value High Byte			

**Table 6-18.19 Command Message Type 1A hex Position Controller Supervisor Attribute - Optional**

<b>Byte</b>	<b>Bit 7</b>	<b>Bit 6</b>	<b>Bit 5</b>	<b>Bit 4</b>	<b>Bit 3</b>	<b>Bit 2</b>	<b>Bit 1</b>	<b>Bit 0</b>
0	Enable	Reg Arm	Hard Stop	Smooth Stop	Direction (V. Mode)	Incremental	Start Block	Load Data/ Start Profile
1					Position Controller Supervisor Attribute to Get			
2	Command Axis Number				Command Message Type			
3					Position Controller Supervisor Attribute to Set			
4					Position Controller Supervisor Attribute Value Low Byte			
5					Position Controller Supervisor Attribute Value Low Middle Byte			
6					Position Controller Supervisor Attribute Value High Middle Byte			
7					Position Controller Supervisor Attribute Value High Byte			

**Position Controller Device, Type: 10<sub>Hex</sub>****Table 6-18.20 Command Message Type 1B hex Position Controller Attribute - Optional**

<b>Byte</b>	<b>Bit 7</b>	<b>Bit 6</b>	<b>Bit 5</b>	<b>Bit 4</b>	<b>Bit 3</b>	<b>Bit 2</b>	<b>Bit 1</b>	<b>Bit 0</b>
0	Enable	Reg Arm	Hard Stop	Smooth Stop	Direction (V. Mode)	Incremental	Start Block	Load Data/ Start Profile
1					Position Controller Attribute to Get			
2		Command Axis Number				Command Message Type		
3					Position Controller Attribute to Set			
4					Position Controller Attribute Value Low Byte			
5					Position Controller Attribute Value Low Middle Byte			
6					Position Controller Attribute Value High Middle Byte			
7					Position Controller Attribute Value High Byte			

**Table 6-18.21 Command Message Type 1C hex Block Sequencer Attribute - Optional**

<b>Byte</b>	<b>Bit 7</b>	<b>Bit 6</b>	<b>Bit 5</b>	<b>Bit 4</b>	<b>Bit 3</b>	<b>Bit 2</b>	<b>Bit 1</b>	<b>Bit 0</b>
0	Enable	Reg Arm	Hard Stop	Smooth Stop	Direction (V. Mode)	Incremental	Start Block	Load Data/ Start Profile
1					Block Sequencer Attribute to Get			
2		Command Axis Number				Command Message Type		
3					Block Sequencer Attribute to Set			
4					Block Sequencer Attribute Value Low Byte			
5					Block Sequencer Attribute Value Low Middle Byte			
6					Block Sequencer Attribute Value High Middle Byte			
7					Block Sequencer Attribute Value High Byte			

**Table 6-18.22 Command Message Type 1D hex Drive Data Attribute - Optional**

<b>Byte</b>	<b>Bit 7</b>	<b>Bit 6</b>	<b>Bit 5</b>	<b>Bit 4</b>	<b>Bit 3</b>	<b>Bit 2</b>	<b>Bit 1</b>	<b>Bit 0</b>
0	Enable	Reg Arm	Hard Stop	Smooth Stop	Direction (V. Mode)	Incremental	Start Block	Load Data/ Start Profile
1					Drive Data Attribute to Get			
2		Command Axis Number				Command Message Type		
3					Drive Data Attribute to Set			
4					Drive Data Attribute Value Low Byte			
5					Drive Data Attribute Value Low Middle Byte			
6					Drive Data Attribute Value High Middle Byte			
7					Drive Data Attribute Value High Byte			

**Table 6-18.23 Command Message Type 1E hex Command Block Attribute - Optional**

<b>Byte</b>	<b>Bit 7</b>	<b>Bit 6</b>	<b>Bit 5</b>	<b>Bit 4</b>	<b>Bit 3</b>	<b>Bit 2</b>	<b>Bit 1</b>	<b>Bit 0</b>
0	Enable	Reg Arm	Hard Stop	Smooth Stop	Direction (V. Mode)	Incremental	Start Block	Load Data/ Start Profile
1					Command Block Attribute to Get/Set			
2		Command Axis Number				Command Message Type		
3					Command Block Instance to Get/Set			
4					Command Block Attribute Value Low Byte			
5					Command Block Attribute Value Low Middle Byte			
6					Command Block Attribute Value High Middle Byte			
7					Command Block Attribute Value High Byte			

**Position Controller Device, Type: 10<sub>Hex</sub>****Table 6-18.24 Command Message Type 1F hex Parameter - Optional**

<b>Byte</b>	<b>Bit 7</b>	<b>Bit 6</b>	<b>Bit 5</b>	<b>Bit 4</b>	<b>Bit 3</b>	<b>Bit 2</b>	<b>Bit 1</b>	<b>Bit 0</b>				
0	Enable	Reg Arm	Hard Stop	Smooth Stop	Direction (V. Mode)	Incremental	Start Block	Load Data/ Start Profile				
1	Parameter Instance to Get											
2	Command Axis Number				Command Message Type							
3	Parameter Instance to Set											
4	Parameter Value Low Byte											
5	Parameter Value Low Middle Byte											
6	Parameter Value High Middle Byte											
7	Parameter Value High Byte											

**6-18.5.2 Semantics****6-18.5.2.1 Load Data/ Start Profile**

Set from zero to one to load command data. The transition of this bit from zero to one will also start a Profile Move when the command message type contained in the command message field is the message type that starts a Profile Move for the mode selected. Refer to Section 6-18.4.3 for an explanation of what commands start a Profile Move for a given mode.

**6-18.5.2.2 Start Block**

This bit is used to execute a Command block or Command block chain. Set from zero to one to execute a command block or command block chain.

**6-18.5.2.3 Incremental**

This bit is used to define the position value as either absolute or incremental. 0 = absolute position value and 1 = incremental position value.

**6-18.5.2.4 Direction (V. Mode)**

This bit is used to control the direction of the motor in Velocity mode. A 1 = forward, positive and a 0 = reverse, negative.

**6-18.5.2.5 Smooth Stop**

This bit is used to bring the motor to a controlled stop at the currently implemented deceleration rate.

**6-18.5.2.6 Hard Stop**

This bit is used to bring the motor to an immediate stop.

**6-18.5.2.7 Reg Arm**

This bit is used to arm the registration input. When the registration input is triggered, the registration action will be executed.

**6-18.5.2.8 Enable**

This bit is used to control the enable output. Clearing this bit will set the enable output inactive and the currently executing motion profile will be aborted.

**6-18.5.2.9 Block #**

This byte defines the block number to be executed when the Start Block bit transitions from zero to one.

**6-18.5.2.10 Command Message Type**

This field defines the Command Message Type

**6-18.5.2.11 Response Message Type**

This field defines the Response Message Type

**6-18.5.2.12 Command Axis Number**

These three bits define the Consumed Axis Connection attribute of the Position Controller Supervisor class. This attribute value specifies the instance number of all of the axis objects whose data are contained in the I/O command message. The command axis number is specified as shown in the table below:

**Table 6-18.25 Command Axis Numbers**

Command Axis Number	Byte 2		
	Bit7	Bit 6	Bit 5
1	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

Note that axis 1 can be specified by either 0 or 1. Axis zero is not allowed.

**6-18.5.2.13 Target Position - Command Message Type 01 hex**

This double word defines the Profile Move's Target Position in position units, when the Load Data /Start Profile bit transitions from zero to one.

**6-18.5.2.14 Target Velocity - Command Message Type 02 hex**

This double word defines the Profile Move's Target Velocity in profile units, when the Load Data /Start Profile bit transitions from zero to one

**6-18.5.2.15 Acceleration - Command Message Type 03 hex**

This double word defines the Profile Move's Acceleration in profile units, when the Load Data /Start Profile bit transitions from zero to one..

**6-18.5.2.16 Deceleration - Command Message Type 04 hex**

This double word defines the Profile Move's Target Position in profile units, when the Load Data /Start Profile bit transitions from zero to one

**6-18.5.2.17 Torque - Command Message Type 05 hex**

This double word is used to set the output torque, when the Load Data /Start Profile bit transitions from zero to one. The torque value will only take effect when in torque mode.  
(Position Controller Object Attribute 3 = 2)

**6-18.5.2.18 Attribute Value – Command Message Types 19 – 1E hex**

This double word defines the value of the attribute to set, when the Load Data/Start Profile bit transitions from zero to one.

**6-18.5.2.19 Object Attribute to Get – Command Message Types 19 – 1D hex**

This byte defines the object attribute to get the value of and return in the response message.

**6-18.5.2.20 Object Attribute to Set – Command Message Types 19 – 1D hex**

This byte defines the object attribute to set to the new value defined by the Attribute Value when the Load Data/Start Profile bit transitions from zero to one.

**Position Controller Device, Type: 10<sub>Hex</sub>****6-18.5.2.21 Command Block Attribute to Get/Set – Command Message Type 1E hex**

This byte defines the attribute of the Command Block Object Instance to get/set, when the Load Data/Start Profile bit transitions from zero to one.

**6-18.5.2.22 Command Block Instance to Get/Set – Command Message Type 1E hex**

This byte defines the instance of the Command Block Object for which the attribute is being get/set, when the Load Data/Start Profile bit transitions from zero to one.

**6-18.5.2.23 Parameter Instance to Get - Command Message Type 1F hex**

This byte defines the instance of the parameter object to get the value of and return in the response message.

**6-18.5.2.24 Parameter Instance to Set - Command Message Type 1F hex**

This byte defines the instance of the parameter object to set to the new value defined by the Parameter Value, when the Load Data/Start Profile bit transitions from zero to one.

**6-18.5.2.25 Parameter Value - Command Message Type 1F hex**

This double word defines the value of the parameter to set, when the Load Data/Start Profile bit transitions from zero to one.

**6-18.5.3 Response Message Types**

The response message type is defined by byte 03 of the message. Bytes 00, 02 and 03 are the same for all response message types. Bytes 01 and 04 through 07 are defined by the Response Message Type code in byte 03.

**Table 6-18.26 Response Message Type 01 hex Actual Position - Optional**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0				
0	Enable	Reg Level	Home Level	Current Direction	General Fault	On Target Position	Block in execution	Profile in Progress				
1	Executing Block Number											
2	Load Complete	Block Fault	FE Fault	Negative Limit	Positive Limit	Rev Limit	Fwd Limit	Fault Input Fault				
3	Response Axis Number				Response Message Type							
4	Actual Position Low Byte											
5	Actual Position Low Middle Byte											
6	Actual Position High Middle Byte											
7	Actual Position High Byte											

**Table 6-18.27 Response Message Type 02 hex Command Position - Optional**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0				
0	Enable	Reg Level	Home Level	Current Direction	General Fault	On Target Position	Block in execution	Profile in Progress				
1	Executing Block Number											
2	Load Complete	Block Fault	FE Fault	Negative Limit	Positive Limit	Rev Limit	Fwd Limit	Fault Input Fault				
3	Response Axis Number				Response Message Type							
4	Commanded Position Low Byte											
5	Commanded Position Low Middle Byte											
6	Commanded Position High Middle Byte											
7	Commanded Position High Byte											

**Position Controller Device, Type: 10<sub>Hex</sub>****Table 6-18.28 Response Message Type 03 hex Actual Velocity - Optional**

<b>Byte</b>	<b>Bit 7</b>	<b>Bit 6</b>	<b>Bit 5</b>	<b>Bit 4</b>	<b>Bit 3</b>	<b>Bit 2</b>	<b>Bit 1</b>	<b>Bit 0</b>
0	Enable	Reg Level	Home Level	Current Direction	General Fault	On Target Position	Block in execution	Profile in Progress
1				Executing Block Number				
2	Load Complete	Block Fault	FE Fault	Negative Limit	Positive Limit	Rev Limit	Fwd Limit	Fault Input Fault
3	Response Axis Number			Response Message Type				
4				Actual Velocity Low Byte				
5				Actual Velocity Low Middle Byte				
6				Actual Velocity High Middle Byte				
7				Actual Velocity High Byte				

**Table 6-18.29 Response Message Type 04 hex Command Velocity - Optional**

<b>Byte</b>	<b>Bit 7</b>	<b>Bit 6</b>	<b>Bit 5</b>	<b>Bit 4</b>	<b>Bit 3</b>	<b>Bit 2</b>	<b>Bit 1</b>	<b>Bit 0</b>
0	Enable	Reg Level	Home Level	Current Direction	General Fault	On Target Position	Block in execution	Profile in Progress
1				Executing Block Number				
2	Load Complete	Block Fault	FE Fault	Negative Limit	Positive Limit	Rev Limit	Fwd Limit	Fault Input Fault
3	Response Axis Number			Response Message Type				
4				Commanded Velocity Low Byte				
5				Commanded Velocity Low Middle Byte				
6				Commanded Velocity High Middle Byte				
7				Commanded Velocity High Byte				

**Table 6-18.30 Response Message Type 05 hex Torque - Optional**

<b>Byte</b>	<b>Bit 7</b>	<b>Bit 6</b>	<b>Bit 5</b>	<b>Bit 4</b>	<b>Bit 3</b>	<b>Bit 2</b>	<b>Bit 1</b>	<b>Bit 0</b>
0	Enable	Reg Level	Home Level	Current Direction	General Fault	On Target Position	Block in execution	Profile in Progress
1				Executing Block Number				
2	Load Complete	Block Fault	FE Fault	Negative Limit	Positive Limit	Rev Limit	Fwd Limit	Fault Input Fault
3	Response Axis Number			Response Message Type				
4				Torque Low Byte				
5				Torque Low Middle Byte				
6				Torque High Middle Byte				
7				Torque High Byte				

**Table 6-18.31 Response Message Type 06 hex Captured Home Position - Optional**

<b>Byte</b>	<b>Bit 7</b>	<b>Bit 6</b>	<b>Bit 5</b>	<b>Bit 4</b>	<b>Bit 3</b>	<b>Bit 2</b>	<b>Bit 1</b>	<b>Bit 0</b>
0	Enable	Reg Level	Home Level	Current Direction	General Fault	On Target Position	Block in execution	Profile in Progress
1				Executing Block Number				
2	Load Complete	Block Fault	FE Fault	Negative Limit	Positive Limit	Rev Limit	Fwd Limit	Fault Input Fault
3	Response Axis Number			Response Message Type				
4				Home Position Low Byte				
5				Home Position Low Middle Byte				
6				Home Position High Middle Byte				
7				Home Position High Byte				

**Position Controller Device, Type: 10<sub>Hex</sub>****Table 6-18.32 Response Message Type 07 hex Captured Index Position - Optional**

<b>Byte</b>	<b>Bit 7</b>	<b>Bit 6</b>	<b>Bit 5</b>	<b>Bit 4</b>	<b>Bit 3</b>	<b>Bit 2</b>	<b>Bit 1</b>	<b>Bit 0</b>
0	Enable	Reg Level	Home Level	Current Direction	General Fault	On Target Position	Block in execution	Profile in Progress
1				Executing Block Number				
2	Load Complete	Block Fault	FE Fault	Negative Limit	Positive Limit	Rev Limit	Fwd Limit	Fault Input Fault
3	Response Axis Number			Response Message Type				
4				Index Position Low Byte				
5				Index Position Low Middle Byte				
6				Index Position High Middle Byte				
7				Index Position High Byte				

**Table 6-18.33 Response Message Type 08 hex Captured Registration Position - Optional**

<b>Byte</b>	<b>Bit 7</b>	<b>Bit 6</b>	<b>Bit 5</b>	<b>Bit 4</b>	<b>Bit 3</b>	<b>Bit 2</b>	<b>Bit 1</b>	<b>Bit 0</b>
0	Enable	Reg Level	Home Level	Current Direction	General Fault	On Target Position	Block in execution	Profile in Progress
1				Executing Block Number				
2	Load Complete	Block Fault	FE FAULT	Negative Limit	Positive Limit	Rev Limit	Fwd Limit	Fault Input Fault
3	Response Axis Number			Response Message Type				
4				Registration Position Low Byte				
5				Registration Position Low Middle Byte				
6				Registration Position High Middle Byte				
7				Registration Position High Byte				

**Table 6-18.34 Response Message Type 14 hex Command/Response Error – Required**

<b>Byte</b>	<b>Bit 7</b>	<b>Bit 6</b>	<b>Bit 5</b>	<b>Bit 4</b>	<b>Bit 3</b>	<b>Bit 2</b>	<b>Bit 1</b>	<b>Bit 0</b>
0	Enable	Reg Level	Home Level	Current Direction	General Fault	On Target Position	Block in execution	Profile in Progress
1				Reserved = 0				
2	Load Complete	Block Fault	FE Fault	Negative Limit	Positive Limit	Rev Limit	Fwd Limit	Fault Input Fault
3	Response Axis Number			Response Message Type				
4				General Error Code				
5				Additional Code				
6				Copy of Command Message Byte 2				
7				Copy of Command Message Byte 3				

**Table 6-18.35 Response Message Type 19 hex Motor Data Attribute - Optional**

<b>Byte</b>	<b>Bit 7</b>	<b>Bit 6</b>	<b>Bit 5</b>	<b>Bit 4</b>	<b>Bit 3</b>	<b>Bit 2</b>	<b>Bit 1</b>	<b>Bit 0</b>
0	Enable	Reg Level	Home Level	Current Direction	General Fault	On Target Position	Block in execution	Profile in Progress
1				Motor Data Attribute to Get				
2	Load Complete	Block Fault	FE Fault	Negative Limit	Positive Limit	Rev Limit	Fwd Limit	Fault Input Fault
3	Response Axis Number			Response Message Type				
4				Motor Data Attribute Value Low Byte				
5				Motor Data Attribute Value Low Middle Byte				
6				Motor Data Attribute Value High Middle Byte				
7				Motor Data Attribute Value High Byte				

**Position Controller Device, Type: 10<sub>Hex</sub>**

**Table 6-18.36 Response Message Type 1A hex Position Controller Supervisor Attribute - Optional**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Enable	Reg Level	Home Level	Current Direction	General Fault	On Target Position	Block in execution	Profile in Progress
1	Position Controller Supervisor Attribute to Get							
2	Load Complete	Block Fault	FE Fault	Negative Limit	Positive Limit	Rev Limit	Fwd Limit	Fault Input Fault
3	Response Axis Number							
4	Response Message Type							
5	Position Controller Supervisor Attribute Value Low Byte							
6	Position Controller Supervisor Attribute Value Low Middle Byte							
7	Position Controller Supervisor Attribute Value High Middle Byte							
	Position Controller Supervisor Attribute Value High Byte							

**Table 6-18.37 Response Message Type 1B hex Position Controller Attribute - Optional**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Enable	Reg Level	Home Level	Current Direction	General Fault	On Target Position	Block in execution	Profile in Progress
1	Position Controller Attribute to Get							
2	Load Complete	Block Fault	FE Fault	Negative Limit	Positive Limit	Rev Limit	Fwd Limit	Fault Input Fault
3	Response Axis Number							
4	Response Message Type							
5	Position Controller Attribute Value Low Byte							
6	Position Controller Attribute Value Low Middle Byte							
7	Position Controller Attribute Value High Middle Byte							
	Position Controller Attribute Value High Byte							

**Table 6-18.38 Response Message Type 1C hex Block Sequencer Attribute - Optional**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Enable	Reg Level	Home Level	Current Direction	General Fault	On Target Position	Block in execution	Profile in Progress
1	Block Sequencer Attribute to Get							
2	Load Complete	Block Fault	FE Fault	Negative Limit	Positive Limit	Rev Limit	Fwd Limit	Fault Input Fault
3	Response Axis Number							
4	Response Message Type							
5	Block Sequencer Attribute Value Low Byte							
6	Block Sequencer Attribute Value Low Middle Byte							
7	Block Sequencer Attribute Value High Middle Byte							
	Block Sequencer Attribute Value High Byte							

**Table 6-18.39 Response Message Type 1D hex Drive Data Attribute - Optional**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Enable	Reg Level	Home Level	Current Direction	General Fault	On Target Position	Block in execution	Profile in Progress
1	Drive Attribute to Get							
2	Load Complete	Block Fault	FE Fault	Negative Limit	Positive Limit	Rev Limit	Fwd Limit	Fault Input Fault
3	Response Axis Number							
4	Response Message Type							
5	Drive Attribute Value Low Byte							
6	Drive Attribute Value Low Middle Byte							
7	Drive Attribute Value High Middle Byte							
	Drive Attribute Value High Byte							

**Position Controller Device, Type: 10<sub>Hex</sub>****Table 6-18.40 Response Message Type 1E hex Command Block Attribute - Optional**

<b>Byte</b>	<b>Bit 7</b>	<b>Bit 6</b>	<b>Bit 5</b>	<b>Bit 4</b>	<b>Bit 3</b>	<b>Bit 2</b>	<b>Bit 1</b>	<b>Bit 0</b>
0	Enable	Reg Level	Home Level	Current Direction	General Fault	On Target Position	Block in execution	Profile in Progress
Command Block Attribute to Get								
2	Load Complete	Block Fault	FE Fault	Negative Limit	Positive Limit	Rev Limit	Fwd Limit	Fault Input Fault
Response Axis Number								
Response Message Type								
4	Command Block Attribute Value Low Byte							
5	Command Block Attribute Value Low Middle Byte							
6	Command Block Attribute Value High Middle Byte							
7	Command Block Attribute Value High Byte							

**Table 6-18.41 Response Message Type 1F hex Parameter - Optional**

<b>Byte</b>	<b>Bit 7</b>	<b>Bit 6</b>	<b>Bit 5</b>	<b>Bit 4</b>	<b>Bit 3</b>	<b>Bit 2</b>	<b>Bit 1</b>	<b>Bit 0</b>
0	Enable	Reg Level	Home Level	Current Direction	General Fault	On Target Position	Block in execution	Profile in Progress
Parameter Instance to Get								
2	Load Complete	Block Fault	FE Fault	Negative Limit	Positive Limit	Rev Limit	Fwd Limit	Fault Input Fault
Response Axis Number								
Response Message Type								
4	Parameter Value Low Byte							
5	Parameter Value Low Middle Byte							
6	Parameter Value High Middle Byte							
7	Parameter Value High Byte							

**6-18.5.4 Semantics****6-18.5.4.1 Profile in Progress**

This bit indicates that a profile move is in progress.

**6-18.5.4.2 Block in Execution**

This bit indicates that a block is in execution. The command block that is currently being executed is returned in byte 1.

**6-18.5.4.3 On Target Position**

This bit indicates whether or not the motor is on the last targeted position. (1 = Current position equals the last target position.)

**6-18.5.4.4 General Fault**

This bit indicates the logical “or” of all fault conditions.

**6-18.5.4.5 Current Direction**

This bit shows the current direction of the motor. If the motor is not moving the bit will indicate the direction of the last commanded move. 0 = reverse or negative direction and 1 = forward or positive direction.

**6-18.5.4.6 Home Level**

This bit reflects the level of the home input.

**6-18.5.4.7 Reg Level**

This bit reflects the level of the registration input.

**6-18.5.4.8 Enable**

This bit indicates the state of the enable output. A 1 indicates the enable output is active.

**6-18.5.4.9 Executing Block #**

This byte defines the currently executing block if the Block In Execution bit is active.

**6-18.5.4.10 Fault Input Fault**

This bit indicates that the fault input is active.

**6-18.5.4.11 Fwd Limit**

This bit indicates that the forward input is active.

**6-18.5.4.12 Rev Limit**

This bit indicates that the reverse input is active.

**6-18.5.4.13 Positive Limit**

This bit indicates that the motor has attempted to travel past the programmed positive limit position. This bit remains valid until the motor is moved within the limits or the programmed limit value is set greater than the current position.

**6-18.5.4.14 Negative Limit**

This bit indicates that the motor has attempted to travel past the programmed negative limit position. This bit remains valid until the motor is moved within the limits or the programmed limit value is set less than the current position.

**6-18.5.4.15 FE Fault**

This bit indicates that a following error fault has occurred. This fault occurs when the following error, or difference between the commanded and actual position, exceeds the programmed allowable following error.

**6-18.5.4.16 Block Fault**

This bit indicates that a block execution fault has occurred. When this happens block execution and motion will cease. This bit is reset when Block Sequencer Block Fault Code attribute (Block Sequencer class, attribute 5) is read.

**6-18.5.4.17 Load Complete**

This bit indicates that the command data contained in the command message has been successfully loaded into the device.

**6-18.5.4.18 Response Message Type**

This byte defines the Response Message Type

**6-18.5.4.19 Response Axis Number**

These three bits report the Produced Axis Connection attribute of the Position Controller Supervisor class. This attribute value specifies the instance number of all of the axis objects whose data is contained in the I/O response message.

**Position Controller Device, Type: 10<sub>Hex</sub>****Table 6-18.42 Response Axis Numbers**

Response Axis Number	Byte 3		
	Bit 7	Bit 6	Bit 5
1	0	0	0
	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

Note that axis 1 can be reported as either binary 0 or 1. Axis zero is not allowed.

**6-18.5.4.20 Actual Position - Response Message Type 01 hex**

This double word reflects the actual position in position units. If position feedback is not used, this word will report the commanded position.

**6-18.5.4.21 Commanded Position - Response Message Type 02 hex**

This double word reflects the commanded or calculated position in position units.

**6-18.5.4.22 Actual Velocity - Response Message Type 03 hex**

This double word reflects the actual velocity in profile units.

**6-18.5.4.23 Command Velocity - Response Message Type 04 hex**

This double word reflects the commanded or calculated velocity in profile units.

**6-18.5.4.24 Torque - Response Message Type 05 hex**

This double word reflects the torque.

**6-18.5.4.25 Home Position - Response Message Type 06 hex**

This double word reflects the captured home position in position units.

**6-18.5.4.26 Index Position - Response Message Type 07 hex**

This double word reflects the captured index position in position units.

**6-18.5.4.27 Registration Position - Response Message Type 08 hex**

This double word reflects the captured registration position in position units.

**6-18.5.4.28 General Error Code – Response Message Type 14 hex**

This byte identifies an error has been encountered. The table below summarizes specific behavior for the Position Controller Profile. See Appendix B for a complete list of General Error codes.

**Table 6-18.43 Additional Error Codes for Position Controller Profile**

General Error Code	Additional Code	Error Description	Semantics
08 <sub>hex</sub>	01 <sub>hex</sub>	Service Not Supported	Command Message type not supported. Additional code 01 takes precedence over additional code 02. <sup>1</sup>
	02 <sub>hex</sub>	Service Not Supported	Response message type not supported.
05 <sub>hex</sub>	01 <sub>hex</sub>	Path Destination Unknown	Consumed axis number was requested that does not exist in the drive.
	02 <sub>hex</sub>	Path Destination Unknown	A produced axis number was requested that does not exist in the drive.
09 <sub>hex</sub>	FF <sub>hex</sub>	Invalid Attribute Value	Load value is out of range.
0E <sub>hex</sub>	FF <sub>hex</sub>	Attribute not Settable	A request to modify a non-modifiable attribute was received.
13 <sub>hex</sub>	FF <sub>hex</sub>	Not Enough Data	I/O command message contained fewer than 8 bytes.
14 <sub>hex</sub>	FF <sub>hex</sub>	Attribute Not Supported	Attribute specified in request was not supported.

<sup>1</sup> If Response Message Type is supported and Command Message Type is not supported, a General Error Code 08, Additional Code 01 shall be returned.

**6-18.5.4.29 Additional Code – Response Message Type 14 hex**

This byte contains an object/service-specific value that further describes the error condition. If the responding object has no additional information to specify, then the value FF<sub>hex</sub> is placed within this field.

**6-18.5.4.30 Attribute Value – Response Message Types 19 – 1E hex**

This double word reflects the value of the attribute to get.

**6-18.5.4.31 Object Attribute to Get – Response Message Types 19 – 1E hex**

This byte defines the object attribute from which to get the value.

**6-18.5.4.32 Parameter Instance to Get - Response Message Type 1F hex**

This byte defines the instance of the parameter object to get the value of and return in the response message.

**6-18.5.4.33 Parameter Value - Response Message Type 1F hex**

This double word reflects the value of the parameter to get.

## 6-18.6 Mapping I/O Message Data Attribute Components

The following table indicates the I/O Data Attribute mapping for the Position Controller Profile Command Messages.

**Table 6-18.44 I/O Assembly Data Mapping – Command Messages**

Data Component Name	Class		Instance Number	Attribute		Data Type
	Name	#		Name	#	
Load Data/ Start Profile	Position Controller	25 <sub>hex</sub>	1-7	Load Data/ Profile Handshake	11	BOOL
Start Block	Block Sequencer	26 <sub>hex</sub>	1-7	Block Execute	2	BOOL
Incremental	Position Controller	25 <sub>hex</sub>	1-7	Incremental	10	BOOL
Direction	Position Controller	25 <sub>hex</sub>	1-7	Direction	23	BOOL
Smooth Stop	Position Controller	25 <sub>hex</sub>	1-7	Hard Stop	20	BOOL
Hard Stop	Position Controller	25 <sub>hex</sub>	1-7	Hard Stop	21	BOOL
Registration Arm	Position Ctrl Supervisor	24 <sub>hex</sub>	1-7	Arm Registration	21	BOOL
Enable	Position Controller	25 <sub>hex</sub>	1-7	Enable	17	BOOL
Block #	Block Sequencer	26 <sub>hex</sub>	1-7	Block	1	USINT
Command Axis Number	Position Ctrl Supervisor Class	24 <sub>hex</sub>	0	Consumed Axis Number	32	USINT
Command Message Type	Position Ctrl Supervisor	24 <sub>hex</sub>	1-7	Cmd Message Type	6	USINT
Response Axis Number	Position Ctrl Supervisor Class	24 <sub>hex</sub>	1-7	Produced Axis Number	33	USINT
Response Message Type	Position Ctrl Supervisor	24 <sub>hex</sub>	1-7	Rspnc Message Type	7	USINT
Target Position	Position Controller	25 <sub>hex</sub>	1-7	Target Position	6	DINT
Target Velocity	Position Controller	25 <sub>hex</sub>	1-7	Target velocity	7	DINT
Acceleration	Position Controller	25 <sub>hex</sub>	1-7	Acceleration	8	DINT
Deceleration	Position Controller	25 <sub>hex</sub>	1-7	Deceleration	9	DINT
Torque	Position Controller	25 <sub>hex</sub>	1-7	Torque	25	DINT
Parameter Value	Parameter	0F <sub>hex</sub>	1-255	Parameter Value	1	Determined by instance of parameter

**Position Controller Device, Type: 10<sub>Hex</sub>**

The following table indicates the I/O Data Attribute mapping for the Position Controller Profile Response Messages.

**Table 6-18.45 I/O Assembly Data Mapping – Response Messages**

Data Component Name	Class		Instance Number	Attribute		Data Type
	Name	#		Name	#	
Profile in Progress	Position Controller	25 <sub>hex</sub>	1-7	Profile in Progress	11	BOOL
Block in Execution	Block Sequencer	26 <sub>hex</sub>	1-7	Block Execute	2	BOOL
On Target Position	Position Controller	25 <sub>hex</sub>	1-7	On Position	12	BOOL
General Fault	Position Ctrl Supervisor	24 <sub>hex</sub>	1-7	General Fault	5	BOOL
Direction	Position Controller	25 <sub>hex</sub>	1-7	Direction	23	BOOL
Home Level	Position Ctrl Supervisor	24 <sub>hex</sub>	1-7	Home Input Level	16	BOOL
Reg Level	Position Ctrl Supervisor	24 <sub>hex</sub>	1-7	Registration Input Level	22	BOOL
Enable State	Position Controller	25 <sub>hex</sub>	1-7	Enable	17	BOOL
Executing Block #	Block Sequencer	26 <sub>hex</sub>	1-7	Current Block	3	USINT
Fault Input Fault	Position Ctrl Supervisor	24 <sub>hex</sub>	1-7	Fault Input	8	BOOL
Fwd Limit	Position Controller	25 <sub>hex</sub>	1-7	Fwd Limit	50	BOOL
Rev Limit	Position Controller	25 <sub>hex</sub>	1-7	Rev Limit	51	BOOL
Positive Limit	Position Controller	25 <sub>hex</sub>	1-7	Positive Limit Triggered	56	BOOL
Negative Limit	Position Controller	25 <sub>hex</sub>	1-7	Negative Limit Triggered	57	BOOL
FE Fault	Position Controller	25 <sub>hex</sub>	1-7	Following Error Fault	47	BOOL
Block Fault	Block Sequencer	26 <sub>hex</sub>	1-7	Block Fault	4	BOOL
Load Complete	Position Controller	25 <sub>hex</sub>	1-7	Load Data Complete	58	BOOL
Response Axis Number	Position Ctrl Supervisor Class	24 <sub>hex</sub>	0	Produced Axis Number	33	USINT
Response Message Type	Position Ctrl Supervisor	24 <sub>hex</sub>	1-7	Rspnc Message Type	7	USINT
Actual Position	Position Controller	25 <sub>hex</sub>	1-7	Actual Position	13	DINT
Commanded Position	Position Controller	25 <sub>hex</sub>	1-7	Commanded Position	15	DINT
Actual Velocity	Position Controller	25 <sub>hex</sub>	1-7	Actual Velocity	14	DINT
Commanded Velocity	Position Controller	25 <sub>hex</sub>	1-7	Commanded Velocity	16	DINT
Torque	Position Controller	25 <sub>hex</sub>	1-7	Torque	25	DINT
Home Position	Position Ctrl Supervisor	24 <sub>hex</sub>	1-7	Home Position	17	DINT
Index Position	Position Ctrl Supervisor	24 <sub>hex</sub>	1-7	Index Position	18	DINT
Registration Position	Position Ctrl Supervisor	24 <sub>hex</sub>	1-7	Registration Position	24	DINT
Parameter Value	Parameter	0F <sub>hex</sub>	1-255	Parameter Value	1	Determined by instance of parameter

## 6-19 Motor Overload Device

### Device Type: 03 Hex

The Motor Overload device profile is part of a “Hierarchy of Motor Control Devices” that are supported by CIP. This hierarchy includes:

- Contactors, Overloads, and Across the Line Motor Starters
- Softstarters
- AC/DC Drives
- Servo Drives

Devices within this hierarchy use a common Control Supervisor object to control state behavior of the device. Devices within this hierarchy also support a hierarchy of “IO Assembly Instance” definitions which are used to pass control and status information to and from a device. Assembly instances are numbered so that each device type is assigned a range of instance numbers, with higher functionality devices supporting higher instance numbers. Devices within the hierarchy can choose to support some instance numbers that are lower than theirs in the hierarchy. For example, an AC Drive may choose to support some instances that are defined for Across the Line Motor Starters. This makes it easier to interchange drives and starters within a system.

This profile makes Motor Starters of the same device type inter-operable, but not directly interchangeable without doing configuration through a unit’s local interface, a network configuration tool or other means of configuring outside the CIP interface.

### 6-19.1 Object Model

The Object Model in Figure 6-19.2 represents a Motor Overload. The table below indicates:

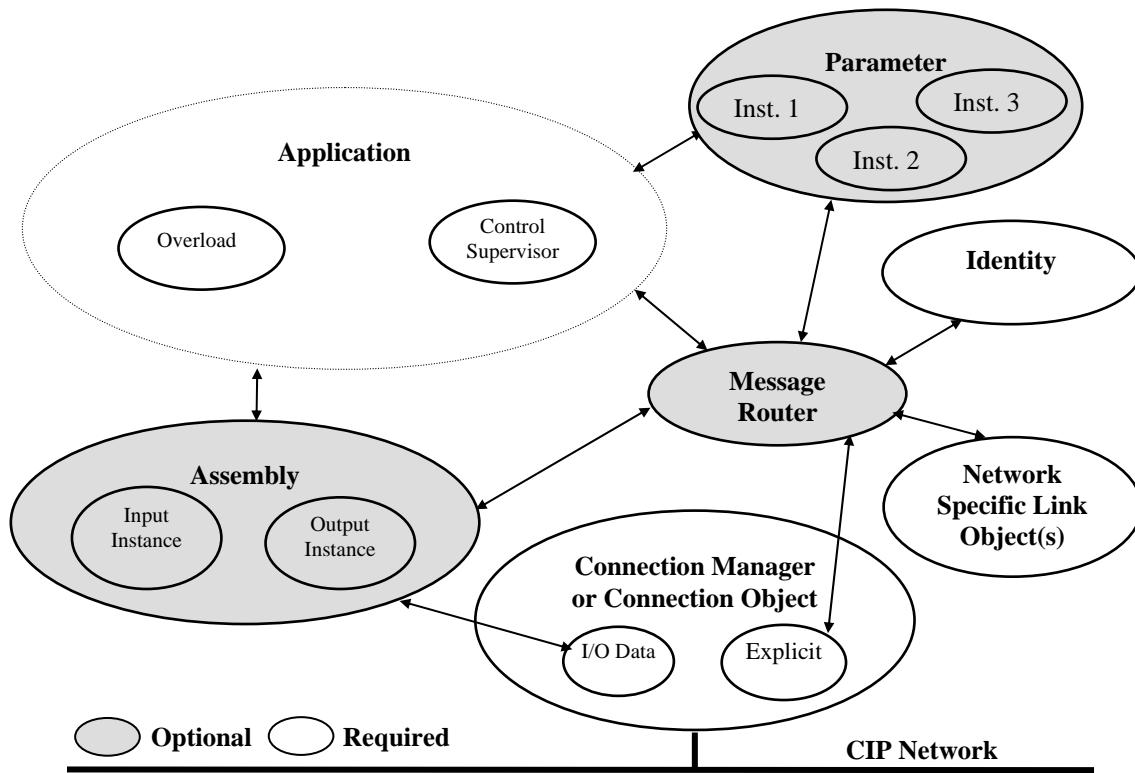
- the object classes present in this device
- whether or not the class is required
- the number of instances present in each class

**Table 6-19.1 Objects Present in a Motor Overload Device**

Object Class	Optional/Required	# of Instances
Message Router	Optional	1
CIP Common Required Objects	Required	See Section 6-2.1
Assembly	Optional	1
Parameter	Optional	-
Control Supervisor	Required	1
Overload	Required	-

Motor Overload Device, Type: 03<sub>Hex</sub>

Figure 6-19.2 Object Model for a Motor Overload Device



## 6-19.2 How Objects Affect Behavior

The objects for this device affect the device's behavior as shown in the table below.

Table 6-19.3 Object Effect on Behavior

Object	Effect on behavior
CIP Common Required	See Section 6-2.2 for details.
Assembly	Defines I/O data format
Parameter	Provides a public interface to device configuration data
Control Supervisor	Manages motor functions and operational states
Overload	Implements overload

### 6-19.3 Defining Object Interfaces

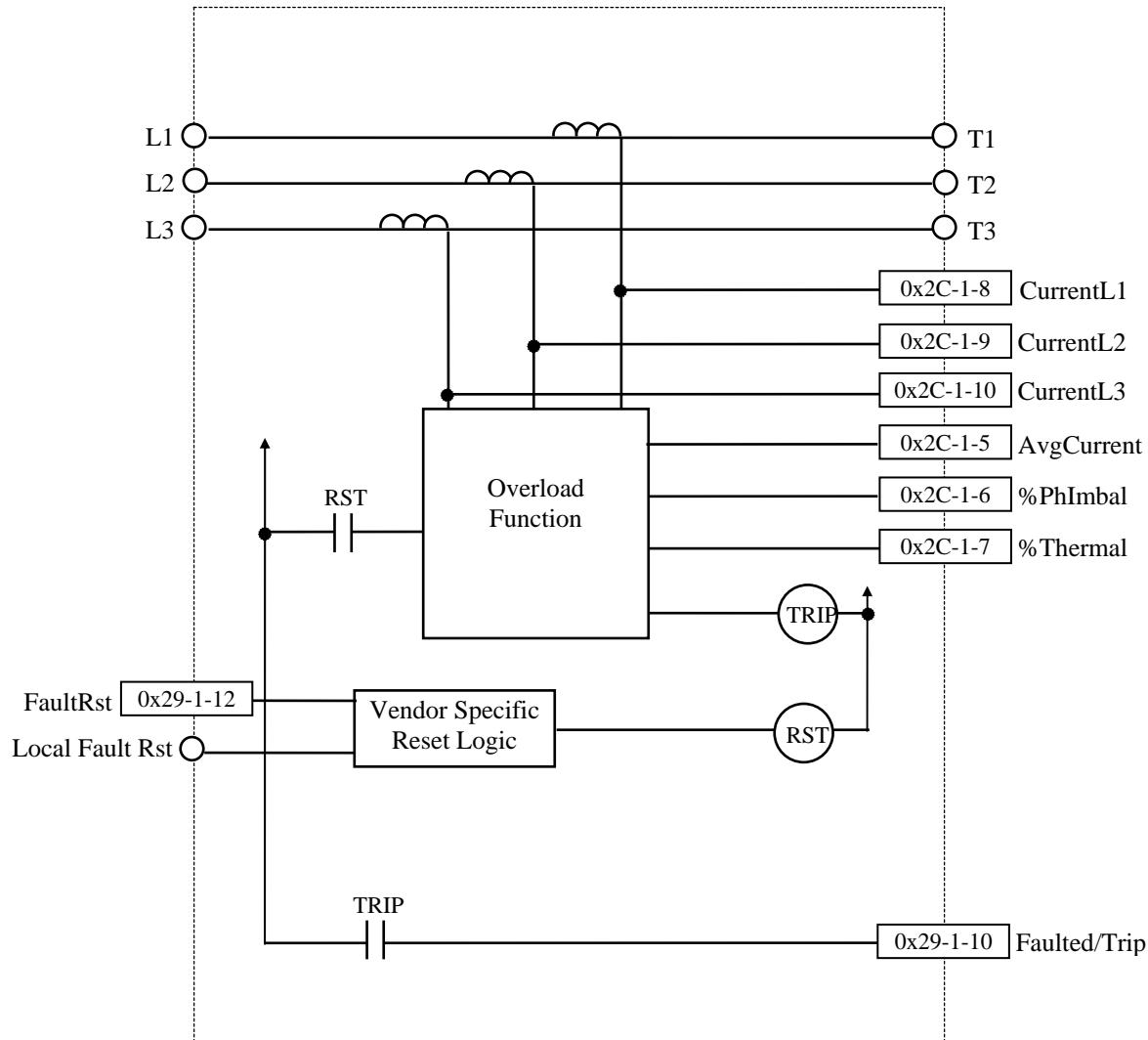
The objects in the Motor Overload have the interfaces listed in the following table:

**Table 6-19.4 Object Interfaces**

Object	Interface
CIP Common Required	See Section 6-2.3 for details.
Assembly	I/O Connection or Message Router
Parameter	Message Router
Control Supervisor	Message Router, Assembly or Parameter Object
Overload	Message Router, Assembly or Parameter Object

### 6-19.4 Contactor Interface and Behavior

**Figure 6-19.5 Behavior Diagram**



## 6-19.5 I/O Assembly Instances

The IO Assembly Instance definitions in this section define the format of the “data” attribute (attribute 3) for IO Assembly Instances. Through the use of predefined instance definitions, IO Assemblies support a hierarchy of motor control devices. The device hierarchy includes motor starters, soft starters, AC and DC drives, and servo drives. Assembly Instances are numbered within the hierarchy so that each device type is assigned a range of Assembly Instance numbers, with higher functionality devices supporting higher instance numbers. **Devices in the hierarchy can choose to support instance numbers that are lower than theirs in the hierarchy.** For example a Softstart may choose to support some IO Assemblies that are defined for Overload. The following table shows the Assembly Instance numbering for the motor control device hierarchy.

**Table 6-19.6 I/O Assembly Instances – Motor Control Device Hierarchy**

Profile	I/O Type	Instance Range
Contactors, Overloads and Starters	Output	1-19
AC/DC Drive	Input	50-69
	Output	20-29
	Input	70-79
Servo Drive	Output	30-49
	Input	80-99

The following IO Assembly Instances are defined for Overloads.

**Table 6-19.7 I/O Assembly Instances - Overloads**

Instance	Type	Name
2	Output	Basic Overload
50	Input	Basic Overload
51	Input	Extended Overload

If a bit is not used in an IO Assembly, it is reserved for use in other Assemblies. Reserved bits in Output Assemblies are ignored by the consuming device. Reserved bits in Input Assemblies are set to zero by the producing device.

### 6-19.5.1 Connection Paths to I/O Assembly Instances

The IO Assembly Instances are chosen for IO Connections by setting the “produced\_connection\_path” (attribute 14) and “consumed\_connection\_path” (attribute 16) attributes in the appropriate connection object.

Motor Control Devices use the Symbolic Segment Type (see Appendix C) to specify paths to the IO Assembly Instances in the Motor Control Hierarchy. IO Assembly Instances are represented by ASCII strings that contain the hex number of the Assembly Instance whose path is to be chosen.

**Motor Overload Device, Type: 03<sub>Hex</sub>**

The following example shows the Symbolic Segment used to specify Output Assembly Instance 20 (14 hex).

**Table 6-19.8 Symbolic Segment Example**

Segment Type			First Character				Second Character		
0	1	1	0	0	0	1	0	0	0
Segment type	Symbol size in bytes (2 bytes)		0	0	1	1	0	0	1
(symbolic)									
			ASCII 1 (31 hex)				ASCII 4 (34 hex)		

**6-19.6 I/O Assembly Data Attribute Format****6-19.6.1 Output Assembly Data Attribute Format****Table 6-19.9 I/O Assembly Instance 2: Basic Overload**

This is the only required output assembly for the device type Motor Overload (03hex)

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Reserved	Reserved	Reserved	Reserved	Reserved	FaultReset	Reserved	Reserved

**6-19.6.2 Input Assembly Data Attribute Format****Table 6-19.10 I/O Assembly Instance 50: Basic Overload**

This is the only required input assembly for the device type Motor Overload.(03hex).

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Reserved	Faulted/Trip						

**Table 6-19.11 I/O Assembly Instance 51: Extended Overload**

This assembly uses some optional attributes

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Warning	Faulted/Trip

**6-19.7 Mapping I/O Assembly Data Attribute Components****6-19.7.1 Mapping for Output Assembly Data Components****Table 6-19.12 Output Assembly Data Mapping**

Data Name	Class Name	Class Number	Instance	Attribute Name	Attribute Number
Fault Reset	Control Supervisor	29	1	FaultRst	12

#### **6-19.7.2 Mapping for Input Assembly Data Components**

**Table 6-19.13 Input Assembly Data Mapping**

Data Name	Class Name	Class Number	Instance	Attribute Name	Attribute Number
Faulted/ Trip	Control Supervisor	0x29	1	Faulted	10
Warning	Control Supervisor	0x29	1	Warning	11

#### **6-19.8 Defining Device Configuration**

Public access to the Control Supervisor Object and the Overload Object must be supported for configuration of a Motor Overload devices. If supported, optional Parameter Objects may be used to access the various configuration attributes in the Control Supervisor Object and the Overload Object.

## **6-20 Weigh Scale Device**

The profile for this device type will be defined by the Open DeviceNet Vendor Association, Inc. and ControlNet International.

## 6-21 Encoder Device

**Device Type: 22 Hex**

### 6-21.1 Introduction

Encoders are used to detect positions of any kind of machines. These devices could be used for following applications: Sensing of angles, distances, tracks, velocity and motion control. This profile covers the measuring principle of absolute and incremental systems as well as the mechanical specification of rotary and linear devices.

### 6-21.2 Object Model

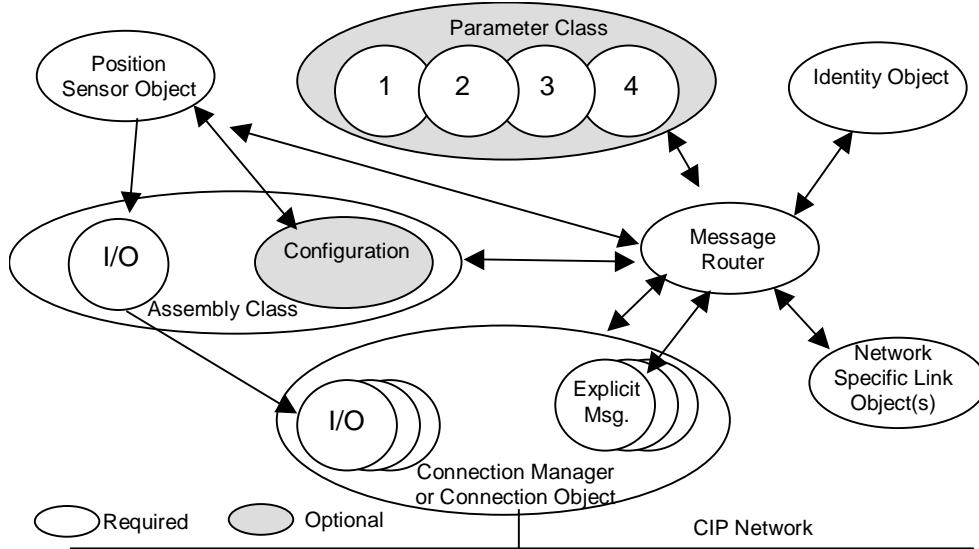
The Object Model in Figure 6-21.2 represents an encoder. The table below indicates:

- The object class
- Whether or not the class is required
- The number of instances present in each class

**Table 6-21.1 Objects Present in an Encoder Device**

Object Class	Optional / Required	# of Instances
CIP Common Required Objects	Required	See Section 6-2.1
Assembly	Required	at least 1 I/O input assembly
Parameter	Optional	-
Position Sensor	Required	1

**Figure 6-21.2 Object Model for an Encoder Device**



### 6-21.3 How Objects Affect Behavior

Objects supported for the encoder affect the device's behavior as shown in the table below.

**Table 6-21.3 Object Effect on Behavior**

Object	Effect on Behavior
CIP Common Required	See Section 6-2.2 for details.
Assembly	Defines I/O data format
Parameter	Provides a public interface to the device's configuration data
Position Sensor	Affects Value (attribute)

### 6-21.4 Defining Object Interfaces

Objects supported for the encoder have the interfaces listed in the table below.

**Table 6-21.4 Objects Interfaces**

Object	Interface
CIP Common Required	See Section 6-2.3 for details.
Assembly	I/O Connection or Message Router
Parameter	Message Router
Position Sensor	Message Router, Assembly Object or Parameter Object

### 6-21.5 I/O Assembly Instances

The following table identifies the I/O Assembly instances, which should be supported by the encoder device.

**Table 6-22.5 I/O Assembly Instances**

Number	Required/Optional	Type	Name
1	Required	Input	Position Value 1 Axis
2	Optional	Input	Position Value 1 axis & Warning Flag 1 axis & Alarm Flag 1 axis
3	Optional	Input	Position Value 1 axis & Velocity 1 axis
4	Optional	Input	Position Value 1 axis & Position Value 2 axis
5	Optional	Input	Position Value 1 axis & Velocity 1 axis & Position Value 2 axis & Velocity 2 axis

## 6-21.6 I/O Assembly Data Attribute Format

The I/O assembly data attributes have the format shown below.

**Table 6-21.6 I/O Assembly Data Attribute Format**

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0			
1	0	Position Value 1 axis (low byte)										
	1	Position Value 1 axis										
	2	Position Value 1 axis										
	3	Position Value 1 axis (high byte)										
2	0	Position Value 1 axis (low byte)										
	1	Position Value 1 axis										
	2	Position Value 1 axis										
	3	Position Value 1 axis (high byte)										
3	4	Vendor Specific			Reserved by CIP			Warning Flag 1 Axis	Alarm Flag 1 Axis			
	0	Position Value 1 axis (low byte)										
	1	Position Value 1 axis										
	2	Position Value 1 axis										
4	3	Position Value 1 axis (high byte)										
	4	Velocity 1 axis (low byte)										
	5	Velocity 1 axis										
	6	Velocity 1 axis										
5	7	Velocity 1 axis (high byte)										
	0	Position Value 1 axis (low byte)										
	1	Position Value 1 axis										
	2	Position Value 1 axis										
6	3	Position Value 1 axis (high byte)										
	4	Velocity 1 axis (low byte)										
	5	Velocity 1 axis										
	6	Velocity 1 axis										
7	7	Velocity 1 axis (high byte)										
	8	Position Value 2 axis (low byte)										
	9	Position Value 2 axis										
	10	Position Value 2 axis										
8	11	Position Value 2 axis (high byte)										
	12	Velocity 2 axis (low byte)										
	13	Velocity 2 axis										
	14	Velocity 2 axis										
9	15	Velocity 2 axis (high byte)										

## 6-21.7 Mapping I/O Assembly Data Attribute Components

The following table indicates the I/O Assembly Data Attribute mapping for the Encoder Profile.

**Table 6-21.7 I/O Assembly Data Mapping**

Data Component Name	Class		Instance Number	Attribute	
	Name	Number		Name	Number
Position Value 1 axis	Position Sensor	23 <sub>hex</sub>	1	Position Value	10
Velocity 1 axis	Position Sensor	23 <sub>hex</sub>	1	Velocity	24
Warning Flag 1 axis	Position Sensor	23 <sub>hex</sub>	1	Warning Flag	49
Alarm Flag 1 axis	Position Sensor	23 <sub>hex</sub>	1	Alarm Flag	46
Position Value 2 axis	Position Sensor	23 <sub>hex</sub>	1	Position Flag	10
Velocity 2 axis	Position Sensor	23 <sub>hex</sub>	1	Velocity	24

## 6-21.8 Defining Device Configuration

Public access to the Position Sensor Object must be supported for configuration of an encoder device. If supported, the optional Parameter Objects may be used to access the various configuration attributes in the Position Sensor Object.

Encoder devices may contain (but are not limited to) any of the Parameter Object instances listed in the table below. Suggested parameter names are also given in the table. The set of parameters instances that are supported by a drive should be numbered sequentially with lower instance numbers assigned to parameters that appear earlier in the table. Vendor specific parameter instances should be numbered sequentially following the instances that appear in the following table.

Parameter Object instances may be implemented as EDS file definitions, parameter stubs, or full parameter objects. See Chapter 5 of the CIP Common specification for a definition of the Parameter Object and an explanation of how it is used for configuration.

## 6-21.9 Mapping Parameter Object Data

The following table indicates the Parameter Object data mapping for encoder devices.

**Table 6-21.8 Parameter Object Data Mapping**

Configuration Parameter Name	Class		Instance Number	Attribute	
	Name	Number		Name	Number
Direct Counting Toggle	Position Sensor	23 <sub>hex</sub>	1	Direction Counting Toggle	12
Commissioning Diagnostic Control	Position Sensor	23 <sub>hex</sub>	1	Commissioning Diagnostic Control	12
Scaling Function Control	Position Sensor	23 <sub>hex</sub>	1	Scaling Function Control	14
Position Format	Position Sensor	23 <sub>hex</sub>	1	Position Format	15
Measuring units per Span	Position Sensor	23 <sub>hex</sub>	1	Measuring Units	16

**Encoder Device, Type: 22<sub>Hex</sub>**

Configuration Parameter Name	Class		Instance Number	Attribute	
	Name	Number		Name	Number
				per Span	
Total Measuring Range in measuring units	Position Sensor	23 <sub>hex</sub>	1	Total Measuring Range in measuring units	17
Position Measuring Increment	Position Sensor	23 <sub>hex</sub>	1	Position Measuring Increment	18
Preset Value	Position Sensor	23 <sub>hex</sub>	1	Preset Value	19
Position Value 1 axis	Position Sensor	23 <sub>hex</sub>	2	Position Value	10
Position Value 2 axis	Position Sensor	23 <sub>hex</sub>	1	Position Value	10
Operating Status	Position Sensor	23 <sub>hex</sub>	1	Operating Status	41
Physical Resolution Span	Position Sensor	23 <sub>hex</sub>	1	Physical resolution span	42
Number of Span	Position Sensor	23 <sub>hex</sub>	1	Number of Span	43
Alarm Flag 1 axis	Position Sensor	23 <sub>hex</sub>	1	Alarm Flag	46
Alarm Flag 2 axis	Position Sensor	23 <sub>hex</sub>	2	Alarm Flag	46
Alarms	Position Sensor	23 <sub>hex</sub>	1	Alarms	44
Supported Alarms	Position Sensor	23 <sub>hex</sub>	1	Supported Alarms	45
Warning Flag 1 axis	Position Sensor	23 <sub>hex</sub>	1	Warning Flag	49
Warning Flag 2 axis	Position Sensor	23 <sub>hex</sub>	2	Warning Flag	49
Warnings	Position Sensor	23 <sub>hex</sub>	1	Warnings	47
Supported Warnings	Position Sensor	23 <sub>hex</sub>	1	Supported Warnings	48

## 6-22 Resolver Device

### Device Type: 09<sub>Hex</sub>

A Resolver mechanically or otherwise detects the absolute position of a shaft. The position information is represented as a binary integer value.

#### 6-22.1 Object Model

The Object Model in figure 6-22.2 represents a resolver. The table below indicates

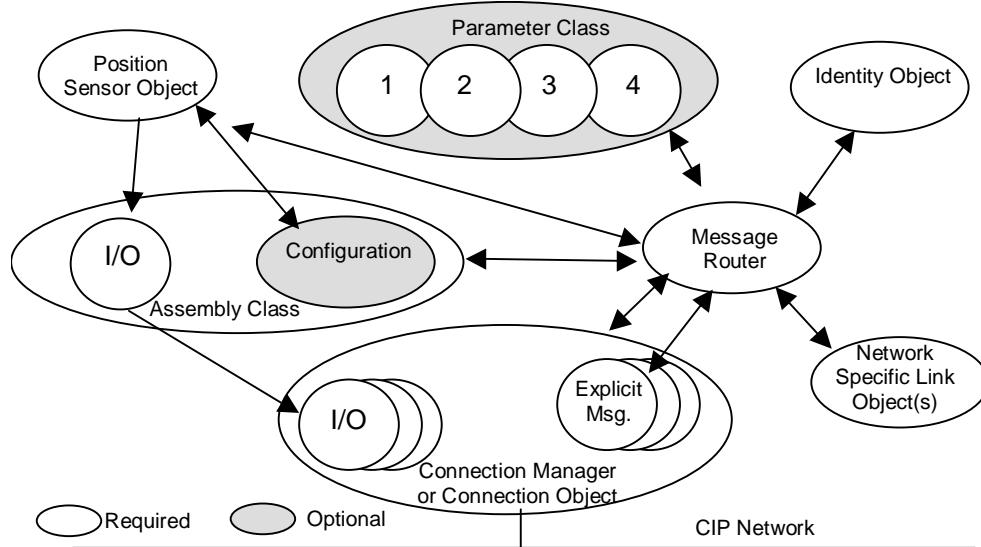
- the object classes present in this device
- whether or not the class is required
- the number of instances present in each class

Chapter 5, The CIP Object Library, provides more details about these objects.

**Table 6-22.1 Objects Present in a Resolver Device**

Object Class	Optional / Required	# of Instances
CIP Common Required Objects	Required	See Section 6-2.1
Assembly	Required	at least 1 I/O input assembly
Parameter	Optional	4
Position Sensor	Required	1

**Figure 6-22.2 Object Model for a Resolver**



## 6-22.2 How Objects Affect Behavior

The objects in this device affect the device's behavior as shown in the following table.

**Table 6-22.3 Object Effect on Behavior**

Object	Effect on Behavior
CIP Common Required	See Section 6-2.2 for details.
Assembly	Defines I/O and/or configuration data format
Parameter	Provides a public interface to the device's configuration data
Position Sensor	Affects Value (attribute), Cam (attribute)

## 6-22.3 Defining Object Interfaces

The objects in this device have the interfaces listed in the following table.

**Table 6-22.4 Objects Interfaces**

Object	Interface
CIP Common Required	See Section 6-2.3 for details.
Assembly	I/O Connection or Message Router
Parameter	Message Router
Position Sensor	Message Router, Assembly Object or Parameter Object

## 6-22.4 I/O Assembly Instances

The following table identifies the I/O assembly instance supported by the Resolver device.

**Table 6-22.5 I/O Assembly Instances**

Number	Required/Optional	Type	Name
1	Optional*	Input	Value
2	Optional*	Input	Value/Cam
3	Optional	Output	SetZero

\*At least 1 input assembly is required

## 6-22.5 I/O Assembly Data Attribute Format

The I/O Assembly Data Attributes have the format shown below.

**Table 6-22.6 I/O Assembly Data Attribute Format**

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	0	Value							
	1								
	2								
	3								
2	0	Value							
	1								
	2								
	3								
	4	Reserved (zero)						CAM	
3	0	Reserved (zero)						SetZero	

## 6-22.6 Mapping I/O Assembly Data Attribute Components

The following table indicates the I/O Assembly Data Attribute mapping for the Resolver device.

**Table 6-22.7 I/O Assembly Data Mapping**

Data Component Name	Class		Instance Number	Attribute	
	Name	Number		Name	Number
Value	Position Sensor	23 <sub>hex</sub>	1	Value	3
CAM	Position Sensor	23 <sub>hex</sub>	1	CAM	4
SetZero	Position Sensor	23 <sub>hex</sub>	1	SetZero	9

## 6-22.7 Configuration Assembly Instances

The following table identifies the configuration assembly instance supported by the Resolver device.

**Table 6-22.8 Configuration Assembly Instances**

Number	Required/Optional	Name
40	Optional	Without CAM
41	Optional	With CAM

## 6-22.8 Configuration Assembly Data Attribute Format

The Configuration Assembly Data Attribute has the format shown below.

**Table 6-22.9 Configuration Assembly Data Format**

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
40	0	Value Bit Resolution							
	1								
	2	Zero Offset							
	3								
	4								
41	0	Value Bit Resolution							
	1								
	2	Zero Offset							
	3								
	4								
	5								
	6	CAM Low Limit							
	7								
	8								
	9								
	10	CAM High Limit							
	11								
	12								

## 6-22.9 Mapping Configuration Assembly Data Attribute Components

The following table indicates the configuration Assembly Data Attribute mapping for the Resolver device.

**Table 6-22.10 Configuration Assembly Data Mapping**

Data Component Name	Class		Instance Number	Attribute	
	Name	Number		Name	Number
Resolution	Position Sensor	23 <sub>hex</sub>	1	Bit Resolution	5
Zero Offset	Position Sensor	23 <sub>hex</sub>	1	Zero Offset	6
CAM Low Limit	Position Sensor	23 <sub>hex</sub>	1	CAM Low	7
CAM High Limit	Position Sensor	23 <sub>hex</sub>	1	CAM High	8

## 6-22.10 Defining Device Configuration

Public access to the Position Sensor Object by the Message Router must be supported for configuration of this device type. If supported, the optional Parameter Object may be used to access the device type's configuration parameters.

If the Parameter Object is supported it must support a minimum of the Parameter Stub attributes, and may optionally support any or all of the Full Parameter Object attributes.

### 6-22.10.1 Parameter Object Instances

The following table indicates the Parameter Object Instances supported by the Resolver device.

**Table 6-22.11 Parameter Object Instances Supported**

Number	Name
1	Value Bit Resolution
2	Zero Offset
3	CAM Low Limit
4	CAM High Limit

### 6-22.10.2 Mapping Parameter Object Data

The following table indicates the Parameter Object data mapping for the Resolver device.

**Table 6-22.12 Parameter Object Data Mapping**

Configuration Parameter Name	Class		Instance Number	Attribute	
	Name	Number		Name	Number
Resolution	Position Sensor	23 <sub>hex</sub>	1	Bit Resolution	5
Zero Offset	Position Sensor	23 <sub>hex</sub>	1	Zero Offset	6
CAM Low Limit	Position Sensor	23 <sub>hex</sub>	1	CAM Low Limit	7
CAM High Limit	Position Sensor	23 <sub>hex</sub>	1	CAM High Limit	8

### 6-22.10.3 Configuration Parameter Definitions

The following sections of an example EDS show the information necessary to define the configuration parameters for a Resolver device.

[ParamClass]	MaxInst=4 Descriptor=0x09 CfgAssembly=2	\$Max Instances \$Parameter Class Descriptor \$Configuration Assembly Instance
[Params]		
Param1=	0, 6, "20 23 24 01 30 05", 0x0000, 8, 1, "Bit Resolution", "Bits", "", 1, 32, (vendor specific), 0, 0, 0, 0, 0, 0, 0, 0;	\$Resolution parameter \$Data placeholder \$Path size and path to attribute \$Descriptor \$Data type and size (USINT) \$Name \$Units \$(not used) \$Min, max and default values \$(not used)
Param2=	0, 6, "20 23 24 01 30 06", 0x0000, 9, 4, "Zero Offset", "", "", 0, 0xFFFFFFFF, 0, 0, 0, 0, 0, 0, 0, 0, 0;	\$Zero Offset parameter \$Data placeholder \$Path size and path to attribute \$Descriptor \$Data type and size (UDINT) \$Name \$Units (none) \$(not used) \$Min, max and default values \$(not used)
Param3=	0, 6, "20 23 24 01 30 07", 0x0000, 9, 4, "CAM Low Limit", "", "", 0, 0xFFFFFFFF, 0, 0, 0, 0, 0, 0, 0, 0, 0;	\$CAM Low Limit \$Data placeholder \$Path size and path to attribute \$Descriptor \$Data type and size (UDINT) \$Name \$Units (none) \$(not used) \$Min, max and default values \$(not used)
Param4=	0, 6, "20 23 24 01 30 08", 0x0000, 9, 4, "CAM High Limit", "", "", 0, 0xFFFFFFFF, 0, 0, 0, 0, 0, 0, 0, 0, 0;	\$CAM High Limit \$Data placeholder \$Path size and path to attribute \$Descriptor \$Data type and size (UDINT) \$Name \$Units (none) \$(not used) \$Min, max and default values \$(not used)

### 6-22.11 Effect of Configuration Parameters on Behavior

The configuration parameters affect the device's behavior as shown below.

**Table 6-22.13 Configuration Parameter Effect on Behavior**

Parameter	Effect on behavior
Bit Resolution	Sets the number of significant bits in the Value Attribute of the Position Sensor Object
Zero Offset	Sets the zero point for the Value Attribute of the Position Sensor Object
CAM Low	Sets the low threshold for the CAM Attribute of the Position Sensor Object
CAM High	Sets the high threshold for the CAM Attribute of the Position Sensor Object

## **6-23      Control Station Device**

The profile for this device type will be defined by the Open DeviceNet Vendor Association, Inc. and ControlNet International.

## **6-24      Message Display Device**

The profile for this device type will be defined by the Open DeviceNet Vendor Association, Inc. and ControlNet International.

## **6-25 Circuit Breaker**

The profile for this device type will be defined by the Open DeviceNet Vendor Association, Inc. and ControlNet International.

## 6-26 Pneumatic Valve Device

### Device Type 1B<sub>hex</sub>

This Device Profile defines minimum requirements for a Pneumatic Valve Manifold with one or more solenoid points and ability to support optional discrete input points.

#### 6-26.1 Object Model

The Object Model is illustrated in Figure 6-26.2 and the object classes are described in the following table:

**Table 6-26.1 Objects Present in a Pneumatic Valve Device**

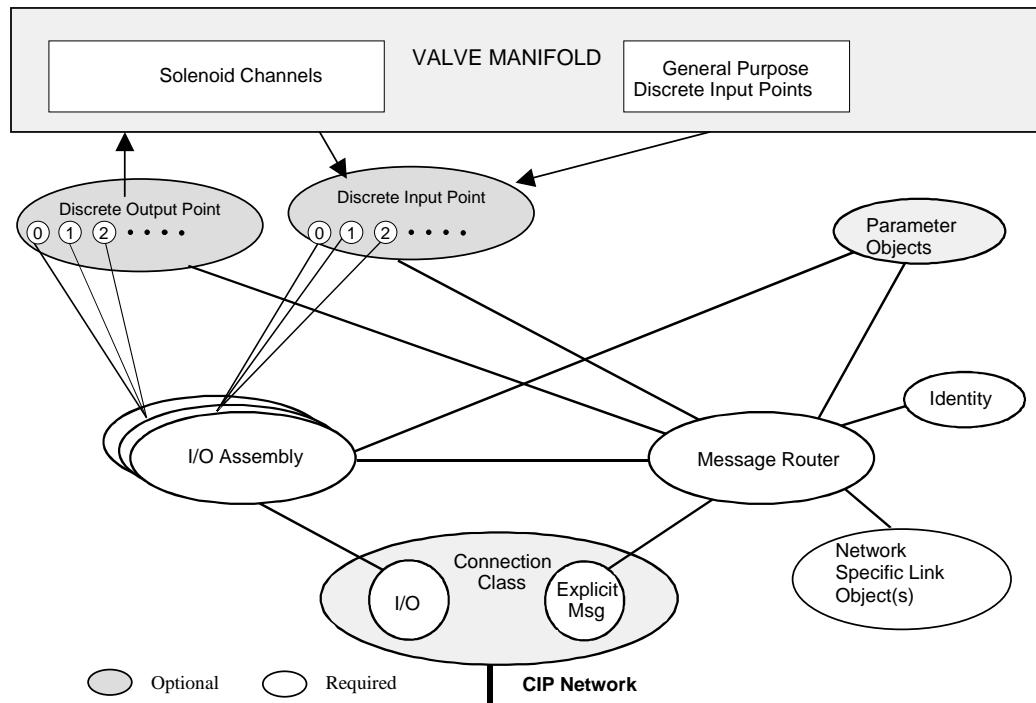
Object Classes	Class ID	Required / Optional	# of Instances
CIP Common Required Objects			See Section 6-2.1
Connection or Connection Manager		Required	2 or more *
Assembly	0x04	Required	1 or more *
Discrete Input **	0x08	Optional	*
Discrete Output ***	0x09	Required	1 or more *
Parameter	0x0F	Optional	Vendor Specific

\* Depends on the level of I/O support provided by the product

\*\* Discrete Input Class includes optional solenoid status points and/or optional discrete input points.

\*\*\* Discrete Output Class includes the solenoid valve points.

**Figure 6-26.2 Object Model for a Pneumatic Valve Manifold**



## 6-26.2 How Objects Affect Behavior

The objects for this device affect its behavior in the following manner:

**Table 6-26.3 Object Effect on Behavior**

Object Classes	Effect on Behavior
Identity	Supports the Reset Service
Message Router	No affect
DeviceNet	Configures port attributes
Assembly	Defines I/O data format
Connection	Contains the number of logical ports
Discrete Input	Defines the behavior of solenoid status and discrete input points
Discrete Output	Defines the behavior of solenoid output points
Parameter	Provides a public interface to device configuration data

## 6-26.3 Defining Object Interfaces

**Table 6-26.4 Object Interfaces**

Object Classes	Interface(s)
CIP Common Required	See Section 6-2.3 for details.
Assembly	Message Router or I/O Connection
Discrete Input	Message Router or Assembly Object
Discrete Output	Message Router or Assembly Object
Parameter	Message Router or Assembly Object

## 6-26.4 I/O Assembly Instances

The following table defines the **Solenoid Status (Input type)** assembly instances:

**Table 6-26.5 I/O Assembly Instances – Solenoid Status**

Instance Number	Type	Name
1	Input	1 Solenoid Status
2	Input	2 Solenoid Status Points
3	Input	4 Solenoid Status Points
4	Input	8 Solenoid Status Points
5	Input	16 Solenoid Status Points
6	Input	24 Solenoid Status Points
7	Input	32 Solenoid Status Points
8	Input	48 Solenoid Status Points
9	Input	64 Solenoid Status Points
10	Input	L Solenoid Status Points

**Pneumatic Valve Device, Type: 1B<sub>Hex</sub>**

The following table defines the **Discrete Input (Input type)** assembly instances:

**Table 6-26.6 I/O Assembly Instances – Discrete Input**

Instance Number	Type	Name
11	Input	1 Discrete Input Point
12	Input	2 Discrete Input Points
13	Input	4 Discrete Input Points
14	Input	8 Discrete Input Points
15	Input	16 Discrete Input Points
16	Input	24 Discrete Input Points
17	Input	32 Discrete Input Points
18	Input	48 Discrete Input Points
19	Input	64 Discrete Input Points
20	Input	M Discrete Input Points

The following table defines the Input type instances where **Solenoid Status points** and **Discrete Input** points are both present.

**Table 6-26.7 I/O Assembly Instances – Solenoid Status & Discrete Input**

Instance Number	Type	Name
21	Input	1 Solenoid Status Points + 1 Discrete Input Points
22	Input	2 Solenoid Status Points + 2 Discrete Input Points
23	Input	4 Solenoid Status Points + 4 Discrete Input Points
24	Input	8 Solenoid Status Points + 8 Discrete Input Points
25	Input	16 Solenoid Status Points + 16 Discrete Input Points
26	Input	24 Solenoid Status Points + 24 Discrete Input Points
27	Input	32 Solenoid Status Points + 32 Discrete Input Points
28	Input	48 Solenoid Status Points + 48 Discrete Input Points
29	Input	64 Solenoid Status Points + 64 Discrete Input Points
30	---	Reserved by CIP

The following table defines the **Solenoid Output (Output type)** assembly instances:

**Pneumatic Valve Device, Type: 1B<sub>Hex</sub>****Table 6-26.8 I/O Assembly Instances – Solenoid Output**

<b>Instance Number</b>	<b>Type</b>	<b>Name</b>
31	Output	Single Solenoid Output Point
32	Output	2 Solenoid Output Points
33	Output	4 Solenoid Output Points
34	Output	8 Solenoid Output Points
35	Output	16 Solenoid Output Points
36	Output	24 Solenoid Output Points
37	Output	32 Solenoid Output Points
38	Output	48 Solenoid Output Points
39	Output	64 Solenoid Output Points
40	Output	N Solenoid Output Points
41 ~ 99	--	Reserved by CIP
100 ~ 199	--	Vendor Specific

**6-26.5 I/O Assembly Data Attribute Format****The I/O Assembly data attribute for *Solenoids Status Data* is shown below:****Table 6-26.9 I/O Assembly Data Attributes – Solenoid Status Data**

<b>Instance</b>	<b>Byte</b>	<b>Bit 7</b>	<b>Bit 6</b>	<b>Bit 5</b>	<b>Bit 4</b>	<b>Bit 3</b>	<b>Bit 2</b>	<b>Bit 1</b>	<b>Bit 0</b>
1	0	Reserved							Solenoid Status1
2	0	Reserved						Solenoid Status2	Solenoid Status1
3	0	Reserved				Solenoid Status4	Solenoid Status3	Solenoid Status2	Solenoid Status1
4	0	Solenoid Status8	Solenoid Status7	Solenoid Status6	Solenoid Status5	Solenoid Status4	Solenoid Status3	Solenoid Status2	Solenoid Status1
5	0	Solenoid Status8	Solenoid Status7	Solenoid Status6	Solenoid Status5	Solenoid Status4	Solenoid Status3	Solenoid Status2	Solenoid Status1
	1	Solenoid Status16	Solenoid Status15	Solenoid Status14	Solenoid Status13	Solenoid Status12	Solenoid Status11	Solenoid Status10	Solenoid Status9
6	0	Solenoid Status8	Solenoid Status7	Solenoid Status6	Solenoid Status5	Solenoid Status4	Solenoid Status3	Solenoid Status2	Solenoid Status1
	1	Solenoid Status16	Solenoid Status15	Solenoid Status14	Solenoid Status13	Solenoid Status12	Solenoid Status11	Solenoid Status10	Solenoid Status9
	2	Solenoid Status24	Solenoid Status23	Solenoid Status22	Solenoid Status21	Solenoid Status20	Solenoid Status19	Solenoid Status18	Solenoid Status17

**Pneumatic Valve Device, Type: 1B<sub>Hex</sub>**

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
7	0	Solenoid Status8	Solenoid Status7	Solenoid Status6	Solenoid Status5	Solenoid Status4	Solenoid Status3	Solenoid Status2	Solenoid Status1
	1	Solenoid Status16	Solenoid Status15	Solenoid Status14	Solenoid Status13	Solenoid Status12	Solenoid Status11	Solenoid Status10	Solenoid Status9
	2	Solenoid Status24	Solenoid Status23	Solenoid Status22	Solenoid Status21	Solenoid Status20	Solenoid Status19	Solenoid Status18	Solenoid Status17
	3	Solenoid Status32	Solenoid Status31	Solenoid Status30	Solenoid Status29	Solenoid Status28	Solenoid Status27	Solenoid Status26	Solenoid Status25
8	0	Solenoid Status8	Solenoid Status7	Solenoid Status6	Solenoid Status5	Solenoid Status4	Solenoid Status3	Solenoid Status2	Solenoid Status1
	1	Solenoid Status16	Solenoid Status15	Solenoid Status14	Solenoid Status13	Solenoid Status12	Solenoid Status11	Solenoid Status10	Solenoid Status9
	2	Solenoid Status24	Solenoid Status23	Solenoid Status22	Solenoid Status21	Solenoid Status20	Solenoid Status19	Solenoid Status18	Solenoid Status17
	3	Solenoid Status32	Solenoid Status31	Solenoid Status30	Solenoid Status29	Solenoid Status28	Solenoid Status27	Solenoid Status26	Solenoid Status25
	4	Solenoid Status40	Solenoid Status39	Solenoid Status38	Solenoid Status37	Solenoid Status36	Solenoid Status35	Solenoid Status34	Solenoid Status33
	5	Solenoid Status48	Solenoid Status47	Solenoid Status46	Solenoid Status45	Solenoid Status44	Solenoid Status43	Solenoid Status42	Solenoid Status41
9	0	Solenoid Status8	Solenoid Status7	Solenoid Status6	Solenoid Status5	Solenoid Status4	Solenoid Status3	Solenoid Status2	Solenoid Status1
	1	Solenoid Status16	Solenoid Status15	Solenoid Status14	Solenoid Status13	Solenoid Status12	Solenoid Status11	Solenoid Status10	Solenoid Status9
	2	Solenoid Status24	Solenoid Status23	Solenoid Status22	Solenoid Status21	Solenoid Status20	Solenoid Status19	Solenoid Status18	Solenoid Status17
	3	Solenoid Status32	Solenoid Status31	Solenoid Status30	Solenoid Status29	Solenoid Status28	Solenoid Status27	Solenoid Status26	Solenoid Status25
	4	Solenoid Status40	Solenoid Status39	Solenoid Status38	Solenoid Status37	Solenoid Status36	Solenoid Status35	Solenoid Status34	Solenoid Status33
	5	Solenoid Status48	Solenoid Status47	Solenoid Status46	Solenoid Status45	Solenoid Status44	Solenoid Status43	Solenoid Status42	Solenoid Status41
	6	Solenoid Status56	Solenoid Status55	Solenoid Status54	Solenoid Status53	Solenoid Status52	Solenoid Status51	Solenoid Status50	Solenoid Status49
	7	Solenoid Status64	Solenoid Status63	Solenoid Status62	Solenoid Status61	Solenoid Status60	Solenoid Status59	Solenoid Status58	Solenoid Status57
10	0	Solenoid Status8	Solenoid Status7	Solenoid Status6	Solenoid Status5	Solenoid Status4	Solenoid Status3	Solenoid Status2	Solenoid Status1
	•								
	K							Solenoid Status L	Solenoid Status L-1

Pneumatic Valve Device, Type: 1B<sub>Hex</sub>

**6-26.5.2 The I/O Assembly data attribute for *General Purpose Discrete Input Data* is shown below:**

**Table 6-26.10 I/O Assembly Data Attributes – Gen Purpose Discrete Input Data**

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
11	0	Reserved							Discrete Input 1
12	0	Reserved						Discrete Input2	Discrete Input1
13	0	Reserved				Discrete Input4	Discrete Input3	Discrete Input2	Discrete Input1
14	0	Discrete Input8	Discrete Input7	Discrete Input6	Discrete Input5	Discrete Input4	Discrete Input3	Discrete Input2	Discrete Input1
15	0	Discrete Input8	Discrete Input7	Discrete Input6	Discrete Input5	Discrete Input4	Discrete Input3	Discrete Input2	Discrete Input1
	1	Discrete Input6	Discrete Input15	Discrete Input14	Discrete Input13	Discrete Input12	Discrete Input11	Discrete Input10	Discrete Input 9
16	0	Discrete Input8	Discrete Input7	Discrete Input6	Discrete Input5	Discrete Input4	Discrete Input3	Discrete Input2	Discrete Input1
	1	Discrete Input16	Discrete Input15	Discrete Input14	Discrete Input13	Discrete Input12	Discrete Input11	Discrete Input10	Discrete Input 9
	2	Discrete Input24	Discrete Input23	Discrete Input22	Discrete Input21	Discrete Input20	Discrete Input19	Discrete Input18	Discrete Input17
17	0	Discrete Input8	Discrete Input7	Discrete Input6	Discrete Input5	Discrete Input4	Discrete Input3	Discrete Input2	Discrete Input1
	1	Discrete Input16	Discrete Input15	Discrete Input14	Discrete Input13	Discrete Input12	Discrete Input11	Discrete Input10	Discrete Input 9
	2	Discrete Input24	Discrete Input23	Discrete Input22	Discrete Input21	Discrete Input20	Discrete Input19	Discrete Input18	Discrete Input17
	3	Discrete Input32	Discrete Input31	Discrete Input 30	Discrete Input29	Discrete Input28	Discrete Input27	Discrete Input26	Discrete Input25
18	0	Discrete Input8	Discrete Input7	Discrete Input6	Discrete Input5	Discrete Input4	Discrete Input3	Discrete Input2	Discrete Input1
	1	Discrete Input16	Discrete Input15	Discrete Input14	Discrete Input13	Discrete Input12	Discrete Input11	Discrete Input10	Discrete Input 9
	2	Discrete Input24	Discrete Input23	Discrete Input22	Discrete Input21	Discrete Input20	Discrete Input19	Discrete Input18	Discrete Input17
	3	Discrete Input32	Discrete Input31	Discrete Input 30	Discrete Input29	Discrete Input28	Discrete Input27	Discrete Input26	Discrete Input25
	4	Discrete Input40	Discrete Input39	Discrete Input38	Discrete Input37	Discrete Input36	Discrete Input35	Discrete Input34	Discrete Input33
	5	Discrete Input48	Discrete Input47	Discrete Input46	Discrete Input45	Discrete Input44	Discrete Input43	Discrete Input42	Discrete Input41

**Pneumatic Valve Device, Type: 1B<sub>Hex</sub>**

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
19	0	Discrete Input8	Discrete Input7	Discrete Input6	Discrete Input5	Discrete Input4	Discrete Input3	Discrete Input2	Discrete Input1
	1	Discrete Input16	Discrete Input15	Discrete Input14	Discrete Input13	Discrete Input12	Discrete Input11	Discrete Input10	Discrete Input 9
	2	Discrete Input24	Discrete Input23	Discrete Input22	Discrete Input21	Discrete Input20	Discrete Input19	Discrete Input18	Discrete Input17
	3	Discrete Input32	Discrete Input31	Discrete Input30	Discrete Input29	Discrete Input28	Discrete Input27	Discrete Input26	Discrete Input25
	4	Discrete Input40	Discrete Input39	Discrete Input38	Discrete Input37	Discrete Input36	Discrete Input35	Discrete Input34	Discrete Input33
	5	Discrete Input48	Discrete Input 47	Discrete Input46	Discrete Input45	Discrete Input 44	Discrete Input43	Discrete Input42	Discrete Input41
	6	Discrete Input56	Discrete Input55	Discrete Input54	Discrete Input53	Discrete Input 52	Discrete Input51	Discrete Input50	Discrete Input 49
	7	Discrete Input 64	Discrete Input63	Discrete Input62	Discrete Input61	Discrete Input60	Discrete Input 59	Discrete Input58	Discrete Input57
20	0	Discrete Input8	Discrete Input7	Discrete Input6	Discrete Input5	Discrete Input4	Discrete Input3	Discrete Input2	Discrete Input1
	•								
	•								
	K						Discrete Input M	Discrete Input M-1	Discrete Input M-2

**6-26.5.3 The I/O Assembly data attribute for Solenoids Status Data plus Discrete Input Data**
**Table 6-26.11 I/O Assembly Data Attributes – Sol. Status & Discrete Input Data**

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
21	0	Reserved						Discrete Input1	Solenoid Status1
22	0	Reserved					Discrete Input2	Discrete Input1	Solenoid Status2
23	0	Discrete Input4	Discrete Input3	Discrete Input2	Discrete Input1	Solenoid Status4	Solenoid Status3	Solenoid Status2	Solenoid Status1
24	0	Solenoid Status8	Solenoid Status7	Solenoid Status6	Solenoid Status5	Solenoid Status4	Solenoid Status3	Solenoid Status2	Solenoid Status1
	1	Discrete Input8	Discrete Input7	Discrete Input6	Discrete Input5	Discrete Input4	Discrete Input3	Discrete Input2	Discrete Input1
25	0	Solenoid Status8	Solenoid Status7	Solenoid Status6	Solenoid Status5	Solenoid Status4	Solenoid Status3	Solenoid Status2	Solenoid Status1
	1	Solenoid Status16	Solenoid Status15	Solenoid Status14	Solenoid Status13	Solenoid Status12	Solenoid Status11	Solenoid Status10	Solenoid Status9
	2	Discrete Input8	Discrete Input7	Discrete Input6	Discrete Input5	Discrete Input4	Discrete Input3	Discrete Input2	Discrete Input1
	3	Discrete Input16	Discrete Input15	Discrete Input14	Discrete Input13	Discrete Input12	Discrete Input11	Discrete Input10	Discrete Input9

**Pneumatic Valve Device, Type: 1B<sub>Hex</sub>**

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
26	0	Solenoid Status8	Solenoid Status7	Solenoid Status6	Solenoid Status5	Solenoid Status4	Solenoid Status3	Solenoid Status2	Solenoid Status1
	1	Solenoid Status16	Solenoid Status15	Solenoid Status14	Solenoid Status13	Solenoid Status12	Solenoid Status11	Solenoid Status10	Solenoid Status9
	2	Solenoid Status24	Solenoid Status23	Solenoid Status22	Solenoid Status21	Solenoid Status20	Solenoid Status19	Solenoid Status18	Solenoid Status17
	3	Discrete Input8	Discrete Input7	Discrete Input6	Discrete Input5	Discrete Input4	Discrete Input3	Discrete Input2	Discrete Input1
	4	Discrete Input16	Discrete Input15	Discrete Input14	Discrete Input13	Discrete Input12	Discrete Input11	Discrete Input10	Discrete Input9
	5	Discrete Input24	Discrete Input23	Discrete Input22	Discrete Input21	Discrete Input20	Discrete Input19	Discrete Input18	Discrete Input17
27	0	Solenoid Status8	Solenoid Status7	Solenoid Status6	Solenoid Status5	Solenoid Status4	Solenoid Status3	Solenoid Status2	Solenoid Status1
	1	Solenoid Status16	Solenoid Status15	Solenoid Status14	Solenoid Status13	Solenoid Status12	Solenoid Status11	Solenoid Status10	Solenoid Status9
	2	Solenoid Status24	Solenoid Status23	Solenoid Status22	Solenoid Status21	Solenoid Status20	Solenoid Status19	Solenoid Status18	Solenoid Status17
	3	Solenoid Status32	Solenoid Status31	Solenoid Status30	Solenoid Status29	Solenoid Status28	Solenoid Status27	Solenoid Status26	Solenoid Status25
	4	Discrete Input8	Discrete Input7	Discrete Input6	Discrete Input5	Discrete Input4	Discrete Input3	Discrete Input2	Discrete Input1
	5	Discrete Input16	Discrete Input15	Discrete Input14	Discrete Input13	Discrete Input12	Discrete Input11	Discrete Input10	Discrete Input9
	6	Discrete Input24	Discrete Input23	Discrete Input22	Discrete Input21	Discrete Input20	Discrete Input19	Discrete Input18	Discrete Input17
	7	Discrete Input32	Discrete Input31	Discrete Input30	Discrete Input29	Discrete Input28	Discrete Input27	Discrete Input26	Discrete Input25
28	0	Solenoid Status8	Solenoid Status7	Solenoid Status6	Solenoid Status5	Solenoid Status4	Solenoid Status3	Solenoid Status2	Solenoid Status1
	1	Solenoid Status16	Solenoid Status15	Solenoid Status14	Solenoid Status13	Solenoid Status12	Solenoid Status11	Solenoid Status10	Solenoid Status9
	2	Solenoid Status24	Solenoid Status23	Solenoid Status22	Solenoid Status21	Solenoid Status20	Solenoid Status19	Solenoid Status18	Solenoid Status17
	3	Solenoid Status32	Solenoid Status31	Solenoid Status30	Solenoid Status29	Solenoid Status28	Solenoid Status27	Solenoid Status26	Solenoid Status25
	4	Solenoid Status40	Solenoid Status39	Solenoid Status38	Solenoid Status37	Solenoid Status36	Solenoid Status35	Solenoid Status34	Solenoid Status33
	5	Solenoid Status48	Solenoid Status47	Solenoid Status46	Solenoid Status45	Solenoid Status44	Solenoid Status43	Solenoid Status42	Solenoid Status41
	6	Discrete Input8	Discrete Input7	Discrete Input6	Discrete Input5	Discrete Input4	Discrete Input3	Discrete Input2	Discrete Input1
	7	Discrete Input16	Discrete Input15	Discrete Input14	Discrete Input13	Discrete Input12	Discrete Input11	Discrete Input10	Discrete Input9
	8	Discrete Input24	Discrete Input23	Discrete Input22	Discrete Input21	Discrete Input20	Discrete Input19	Discrete Input18	Discrete Input17
	9	Discrete Input32	Discrete Input31	Discrete Input30	Discrete Input29	Discrete Input28	Discrete Input27	Discrete Input26	Discrete Input25

**Pneumatic Valve Device, Type: 1B<sub>Hex</sub>**

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	10	Discrete Input40	Discrete Input39	Discrete Input38	Discrete Input37	Discrete Input36	Discrete Input35	Discrete Input34	Discrete Input33
	11	Discrete Input48	Discrete Input 47	Discrete Input46	Discrete Input45	Discrete Input 44	Discrete Input43	Discrete Input42	Discrete Input41
29	0	Solenoid Status8	Solenoid Status7	Solenoid Status6	Solenoid Status5	Solenoid Status4	Solenoid Status3	Solenoid Status2	Solenoid Status1
	1	Solenoid Status16	Solenoid Status15	Solenoid Status14	Solenoid Status13	Solenoid Status12	Solenoid Status11	Solenoid Status10	Solenoid Status9
	2	Solenoid Status24	Solenoid Status23	Solenoid Status22	Solenoid Status21	Solenoid Status20	Solenoid Status19	Solenoid Status18	Solenoid Status17
	3	Solenoid Status32	Solenoid Status31	Solenoid Status30	Solenoid Status29	Solenoid Status28	Solenoid Status27	Solenoid Status26	Solenoid Status25
	4	Solenoid Status40	Solenoid Status39	Solenoid Status38	Solenoid Status37	Solenoid Status36	Solenoid Status35	Solenoid Status34	Solenoid Status33
	5	Solenoid Status48	Solenoid Status47	Solenoid Status46	Solenoid Status45	Solenoid Status44	Solenoid Status43	Solenoid Status42	Solenoid Status41
	6	Solenoid Status56	Solenoid Status55	Solenoid Status54	Solenoid Status53	Solenoid Status52	Solenoid Status51	Solenoid Status50	Solenoid Status49
	7	Solenoid Status64	Solenoid Status63	Solenoid Status62	Solenoid Status61	Solenoid Status60	Solenoid Status59	Solenoid Status58	Solenoid Status57
	8	Discrete Input8	Discrete Input7	Discrete Input6	Discrete Input5	Discrete Input4	Discrete Input3	Discrete Input2	Discrete Input 1
	9	Discrete Input16	Discrete Input15	Discrete Input14	Discrete Input13	Discrete Input12	Discrete Input11	Discrete Input10	Discrete Input9
	10	Discrete Input24	Discrete Input23	Discrete Input22	Discrete Input 21	Discrete Input 20	Discrete Input19	Discrete Input18	Discrete Input17
	11	Discrete Input32	Discrete Input31	Discrete Input30	Discrete Input29	Discrete Input28	Discrete Input27	Discrete Input26	Discrete Input25
	12	Discrete Input40	Discrete Input39	Discrete Input38	Discrete Input37	Discrete Input36	Discrete Input35	Discrete Input34	Discrete Input33
	13	Discrete Input48	Discrete Input 47	Discrete Input46	Discrete Input45	Discrete Input 44	Discrete Input43	Discrete Input42	Discrete Input41
	14	Discrete Input56	Discrete Input55	Discrete Input54	Discrete Input53	Discrete Input 52	Discrete Input51	Discrete Input50	Discrete Input 49
	15	Discrete Input 64	Discrete Input63	Discrete Input62	Discrete Input61	Discrete Input60	Discrete Input 59	Discrete Input58	Discrete Input57

Pneumatic Valve Device, Type: 1B<sub>Hex</sub>6-26.5.4 The I/O Assembly data attribute for the *Solenoid Valves Output Data*

Table 6-26.12 I/O Assembly Data Attributes – Solenoid Valves Output Data

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
31	0	Reserved							Solenoid Output1
32	0	Reserved						Solenoid Output2	Solenoid Output1
33	0	Reserved				Solenoid Output4	Solenoid Output3	Solenoid Output2	Solenoid Output1
34	0	Solenoid Output8	Solenoid Output7	Solenoid Output6	Solenoid Output5	Solenoid Output4	Solenoid Output3	Solenoid Output2	Solenoid Output1
35	0	Solenoid Output8	Solenoid Output7	Solenoid Output6	Solenoid Output5	Solenoid Output4	Solenoid Output3	Solenoid Output2	Solenoid Output1
	1	Solenoid Output16	Solenoid Output15	Solenoid Output14	Solenoid Output13	Solenoid Output12	Solenoid Output11	Solenoid Output10	Solenoid Output9
36	0	Solenoid Output8	Solenoid Output7	Solenoid Output6	Solenoid Output5	Solenoid Output4	Solenoid Output3	Solenoid Output2	Solenoid Output1
	1	Solenoid Output16	Solenoid Output15	Solenoid Output14	Solenoid Output13	Solenoid Output12	Solenoid Output11	Solenoid Output10	Solenoid Output9
	2	Solenoid Output24	Solenoid Output23	Solenoid Output22	Solenoid Output21	Solenoid Output20	Solenoid Output19	Solenoid Output18	Solenoid Output17
37	0	Solenoid Output8	Solenoid Output7	Solenoid Output6	Solenoid Output5	Solenoid Output4	Solenoid Output3	Solenoid Output2	Solenoid Output1
	1	Solenoid Output16	Solenoid Output15	Solenoid Output14	Solenoid Output13	Solenoid Output12	Solenoid Output11	Solenoid Output10	Solenoid Output9
	2	Solenoid Output24	Solenoid Output23	Solenoid Output22	Solenoid Output21	Solenoid Output20	Solenoid Output19	Solenoid Output18	Solenoid Output17
	3	Solenoid Output32	Solenoid Output31	Solenoid Output30	Solenoid Output29	Solenoid Output28	Solenoid Output27	Solenoid Output26	Solenoid Output25
38	0	Solenoid Output8	Solenoid Output7	Solenoid Output6	Solenoid Output5	Solenoid Output4	Solenoid Output3	Solenoid Output2	Solenoid Output1
	1	Solenoid Output16	Solenoid Output15	Solenoid Output14	Solenoid Output13	Solenoid Output12	Solenoid Output11	Solenoid Output10	Solenoid Output9
	2	Solenoid Output24	Solenoid Output23	Solenoid Output22	Solenoid Output21	Solenoid Output20	Solenoid Output19	Solenoid Output18	Solenoid Output17
	3	Solenoid Output32	Solenoid Output31	Solenoid Output30	Solenoid Output29	Solenoid Output28	Solenoid Output27	Solenoid Output26	Solenoid Output25
	4	Solenoid Output40	Solenoid Output39	Solenoid Output38	Solenoid Output37	Solenoid Output36	Solenoid Output35	Solenoid Output34	Solenoid Output33
	5	Solenoid Output48	Solenoid Output47	Solenoid Output46	Solenoid Output45	Solenoid Output44	Solenoid Output43	Solenoid Output42	Solenoid Output41

**Pneumatic Valve Device, Type: 1B<sub>Hex</sub>**

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
39	0	Solenoid Output8	Solenoid Output7	Solenoid Output6	Solenoid Output5	Solenoid Output4	Solenoid Output3	Solenoid Output2	Solenoid Output1
	1	Solenoid Output16	Solenoid Output15	Solenoid Output14	Solenoid Output13	Solenoid Output12	Solenoid Output11	Solenoid Output10	Solenoid Output9
	2	Solenoid Output24	Solenoid Output23	Solenoid Output22	Solenoid Output21	Solenoid Output20	Solenoid Output19	Solenoid Output18	Solenoid Output17
	3	Solenoid Output32	Solenoid Output31	Solenoid Output30	Solenoid Output29	Solenoid Output28	Solenoid Output27	Solenoid Output26	Solenoid Output25
	4	Solenoid Output40	Solenoid Output39	Solenoid Output38	Solenoid Output37	Solenoid Output36	Solenoid Output35	Solenoid Output34	Solenoid Output33
	5	Solenoid Output48	Solenoid Output47	Solenoid Output46	Solenoid Output45	Solenoid Output44	Solenoid Output43	Solenoid Output42	Solenoid Output41
	6	Solenoid Output56	Solenoid Output55	Solenoid Output54	Solenoid Output53	Solenoid Output52	Solenoid Output51	Solenoid Output50	Solenoid Output49
	7	Solenoid Output64	Solenoid Output63	Solenoid Output62	Solenoid Output61	Solenoid Output60	Solenoid Output59	Solenoid Output58	Solenoid Output57
40	0	Solenoid Output8	Solenoid Output7	Solenoid Output6	Solenoid Output5	Solenoid Output4	Solenoid Output3	Solenoid Output2	Solenoid Output1
	•								
	K							Solenoid Output N	Solenoid Output N-1

**6-26.6 Mapping I/O Assembly Data Attribute Components**

The following table indicates the I/O assembly Data attribute mapping for the Pneumatic Valve Manifold device for the input assemblies numbered 1 through 10. These assemblies only present Solenoid Status information. The exact meaning of a “Solenoid Status” point is defined and assigned in the vendor instruction manual. Unless otherwise specified, the value 1=TRUE=“Energized Solenoid” is assumed in each assembly. Likewise, the value 0=FALSE=“De-energized Solenoid” is assumed in each assembly unless otherwise specified.

**Table 6-26.13 I/O Assembly Data Mapping – Input Assemblies 1-10**

Data Component Name	Class		Instance Number	Attribute	
	Name	Number		Name	Number
Solenoid Status L	Discrete Input Point	08 <sub>hex</sub>	L	Value	3

The following table indicates the I/O assembly Data attribute mapping for Pneumatic Valve Manifold device for the input assemblies numbered 11 through 20. These assemblies only present values from General Purpose Discrete Input points. These points are not explicitly associated with Solenoid Value status (as defined in assemblies 1 through 10). Rather, the exact meaning of each point shall be defined and assigned by the vendor instruction manual.

**Pneumatic Valve Device, Type: 1B<sub>Hex</sub>****Table 6-26.14 I/O Assembly Data Mapping – Input Assemblies 11-20**

Data Component Name	Class		Instance Number	Attribute	
	Name	Number		Name	Number
Discrete Input <i>M</i>	Discrete Input Point	08 <sub>hex</sub>	<i>M</i>	Value	3

The following table indicates the I/O assembly Data attribute mapping for the Pneumatic Valve Manifold device for the input assemblies numbered 21 through 29. These assemblies present both Solenoid Status information as well as General Purpose Discrete Input point values. The exact meaning of a “Solenoid Status” point is defined and assigned in the vendor instruction manual. Unless otherwise specified, the value 1=TRUE=“Energized Solenoid” is assumed in each assembly. Likewise, the value 0=FALSE=“De-energized Solenoid” is assumed in each assembly unless otherwise specified. The Discrete Input point values are not explicitly associated with Solenoid Value status. The exact meaning of each point shall be defined and assigned by the vendor instruction manual.

**Table 6-26.15 I/O Assembly Data Mapping – Input Assemblies 21-29**

Data Component Name	Class		Instance Number	Attribute	
	Name	Number		Name	Number
Solenoid Status <i>L</i>	Discrete Input Point	08 <sub>hex</sub>	<i>L</i>	Value	3
Discrete Input <i>M</i>	Discrete Input Point	08 <sub>hex</sub>	<i>M</i>	Value	3

The following table indicates the I/O assembly Data attribute mapping for Pneumatic Valve Manifold device for the Solenoid output assemblies.

**Table 6-26.16 I/O Assembly Data Mapping – Solenoid Output Assemblies**

Data Component Name	Class		Instance Number	Attribute	
	Name	Number		Name	Number
Solenoid Output <i>L</i>	Discrete Input Point	08 <sub>hex</sub>	<i>N</i>	Value	3

**Important:** If I/O Assembly instances 10 or 20 are supported, the “Max Instance” attribute at the class level of the Discrete Input Point class must be supported. If I/O Assembly instance 40 is supported, the “Max Instance” attribute at the class level of the Discrete Output Point class must be supported.

## 6-26.7 Configuration Data and Public Interfaces

Device configuration data and its public interfaces are vendor specific.

## 6-26.8 Parameter Object Instances

The Parameter Object Class, its instances and data mapping are optional and vendor specific.

## 6-27 Contactor Device

### Device Type: 15<sub>hex</sub>

The Contactor device profile is part of a “Hierarchy of Motor Control Devices” that are supported by CIP. This hierarchy includes:

- Contactors, Overloads, and Across the Line Motor Starters
- Softstarters
- AC/DC Drives
- Servo Drives

Devices within this hierarchy use a common Control Supervisor object to control state behavior of the device. Devices within this hierarchy also support a hierarchy of “IO Assembly Instance” definitions which are used to pass control and status information to and from a device. Assembly instances are numbered so that each device type is assigned a range of instance numbers, with higher functionality devices supporting higher instance numbers. Devices within the hierarchy can choose to support some instance numbers that are lower than theirs in the hierarchy. For example, an AC Drive may choose to support some instances that are defined for Across the Line Motor Starters. This makes it easier to interchange drives and starters within a system.

This profile makes Motor Starters of the same device type inter-operable, but not directly interchangeable without doing configuration through a unit’s local interface, a network configuration tool or other means of configuring outside the CIP interface.

### 6-27.1 Object Model

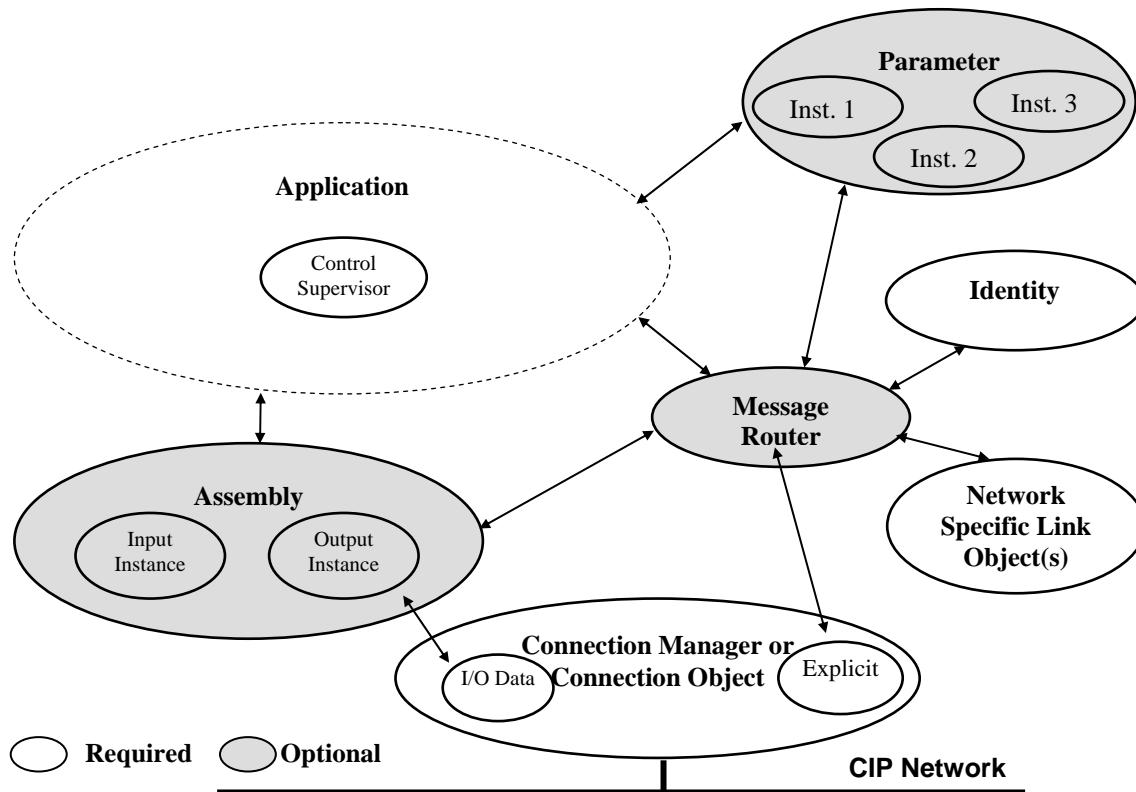
The Object Model in Figure 6-27.2 represents a Contactor. The table below indicates:

- the object classes present in this device
- whether or not the class is required
- the number of instances present in each class

**Table 6-27.1 Objects Present in a Contactor Device**

Object Class	Optional/Required	# of Instances
Message Router	Optional	-
CIP Common Required Objects	Required	See Section 6-2.1
Assembly	Optional	1
Parameter	Optional	-
Control Supervisor	Required	1

Figure 6-27.2 Object Model for a Contactor Device



## 6-27.2 How Objects Affect Behavior

The objects for this device affect the device's behavior as shown in the table below.

Table 6-27.3 Object Effect On Behavior

Object	Effect on behavior
CIP Common Required	See Section 6-2.2 for details.
Assembly	Defines I/O data format
Parameter	Provides a public interface to device configuration data
Control Supervisor	Manages motor functions and operational states

## 6-27.3 Defining Object Interfaces

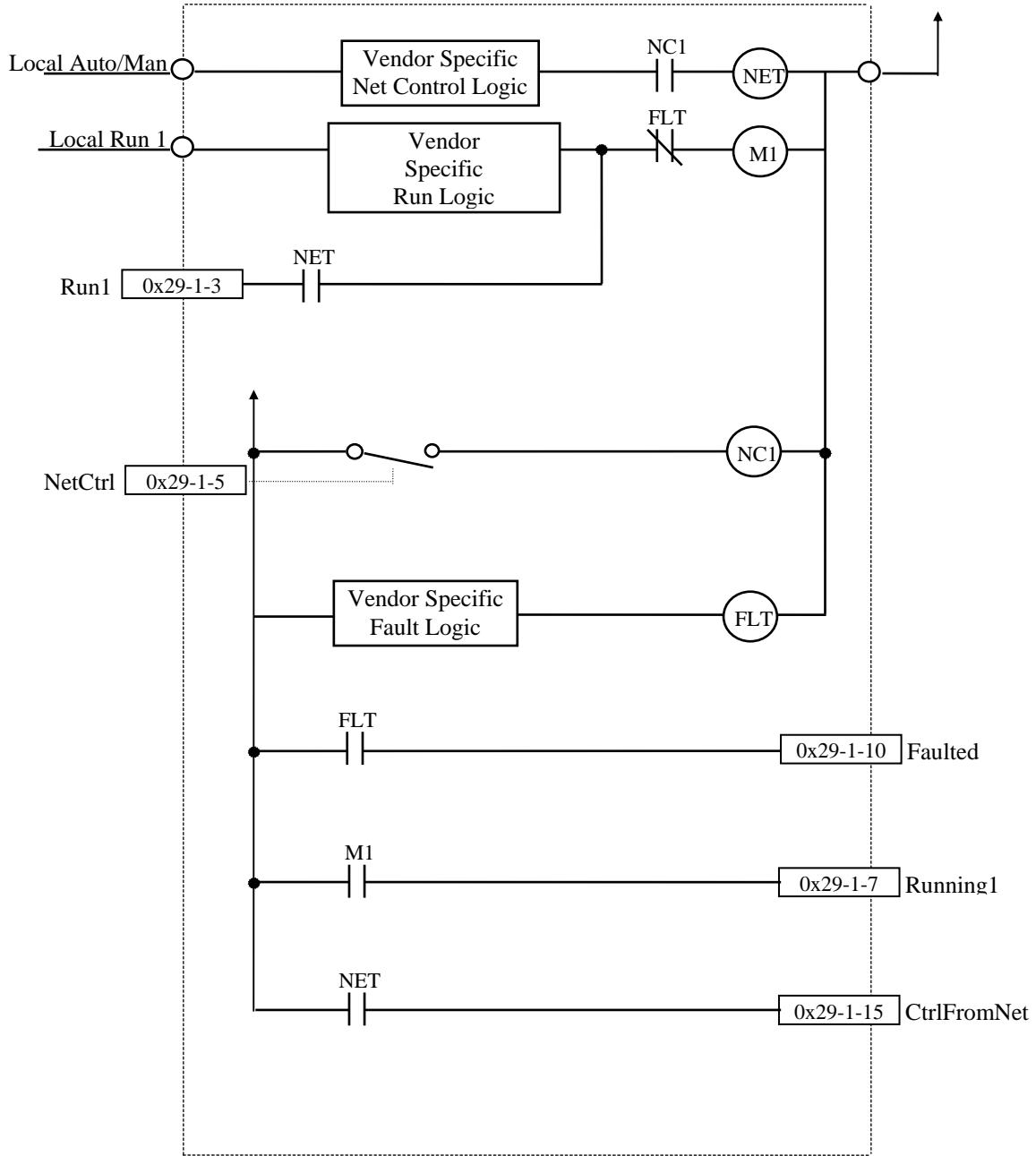
The objects in the Contactor Device have the interfaces listed in the following table:

Table 6-27.4 Object Interfaces

Object	Interface
CIP Common Required	See Section 6-2.3 for details.
Assembly	I/O Connection or Message Router
Parameter	Message Router
Control Supervisor	Message Router, Assembly or Parameter Object

## 6-27.4 Contactor Interface and Behavior

Figure 6-27.5 Object Interfaces and Effect on Behavior



## 6-27.5 I/O Assembly Instance

The IO Assembly Instance definitions in this section define the format of the “data” attribute (attribute 3) for IO Assembly Instances. Through the use of predefined instance definitions, IO Assemblies support a hierarchy of motor control devices. The device hierarchy includes motor starters, soft starters, AC and DC drives, and servo drives. Assembly Instances are numbered within the hierarchy so that each device type is assigned a range of Assembly Instance numbers, with higher functionality devices supporting higher instance numbers. **Devices in the hierarchy can choose to support instance numbers that are lower than theirs in the hierarchy.** For example a Softstart may choose to support some IO Assemblies that are defined for Overload. The following table shows the Assembly Instance numbering for the motor control device hierarchy.

**Table 6-27.6 I/O Assembly Instances – Motor Control Device**

Profile	I/O Type	Instance Range
Contactors, Overloads and Starters	Output	1-19
	Input	50-69
AC/DC Drive	Output	20-29
	Input	70-79
Servo Drive	Output	30-49
	Input	80-99

The following IO Assembly Instances are defined for Contactors.

**Table 6-27.7 I/O Assembly Instances – Contactor Device**

Instance	Type	Name
1	Output	Basic Contactor
4	Output	Extended Contactor

If a bit is not used in an IO Assembly, it is reserved for use in other Assemblies. Reserved bits in Output Assemblies are ignored by the consuming device. Reserved bits in Input Assemblies are set to zero by the producing device.

### 6-27.5.1 Connection Paths to I/O Assembly Instances

The IO Assembly Instances are chosen for IO Connections by setting the “produced\_connection\_path” (attribute 14) and “consumed\_connection\_path” (attribute 16) attributes in the appropriate connection object.

Motor Control Devices use the Symbolic Segment Type (see Appendix C) to specify paths to the IO Assembly Instances in the Motor Control Hierarchy. IO Assembly Instances are represented by ASCII strings that contain the hex number of the Assembly Instance whose path is to be chosen.

The following example shows the Symbolic Segment used to specify Output Assembly Instance 20 (14 hex).

**Figure 6-27.8 Symbolic Segment Example**

Segment Type	First Character	Second Character	
0 1 1   0 0 0 1 0 Segment type symbolic	0 0 1 1 0 0 0 1 Symbol size in bytes (2 bytes)	0 0 1 1 0 1 0 0 ASCII 1 (31 hex)	ASCII 4 (34 hex)

## 6-27.6 I/O Assembly Data Attribute Format

Table 6-27.9 I/O Assembly Data Attribute Format – Instance 1, Basic Contactor

### Instance 1: Basic Contactor

This is the only required output assembly for device types Motor Contactor (0x15hex) and Softstart (0x15hex).

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Reserved	Run1						

Table 6-27.10 I/O Assembly Data Attribute Format – Instance 4, Ext. Contactor

### Instance 4: Extended Contactor (see table for functional assignments)

This assembly uses some optional attributes..

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Run2	Run1

## 6-27.7 Mapping I/O Assembly Data Attribute Components

The following table indicates the I/O Assembly Data Attribute mapping for Contactor Output Assemblies.

Table 6-27.11 I/O Assembly Data Attribute Mapping

Data Name	Class Name	Class Number	Instance	Attribute Name	Attribute Number
Run1	Control Supervisor	0x 29	1	Run1	3
Run2	Control Supervisor	0x 29	1	Run2	4

## 6-27.8 Defining Device Configuration

Public access to the Control Supervisor Object must be supported for configuration of Contactor devices. If supported, optional Parameter Objects may be used to access the various configuration attributes in the Control Supervisor Object.

## 6-28 Motor Starter Device

### Device Type: 16<sub>hex</sub>

The Motor Starter device profile is part of a “Hierarchy of Motor Control Devices” that are supported by CIP. This hierarchy includes:

- Contactors, Overloads, and Across the Line Motor Starters
- Softstarters
- AC/DC Drives
- Servo Drives

Devices within this hierarchy use a common Control Supervisor object to control state behavior of the device. Devices within this hierarchy also support a hierarchy of “IO Assembly Instance” definitions which are used to pass control and status information to and from a device. Assembly instances are numbered so that each device type is assigned a range of instance numbers, with higher functionality devices supporting higher instance numbers. Devices within the hierarchy can choose to support some instance numbers that are lower than theirs in the hierarchy. For example, an AC Drive may choose to support some instances that are defined for Across the Line Motor Starters. This makes it easier to interchange drives and starters within a system.

This profile makes Motor Starters of the same device type inter-operable, but not directly interchangeable without doing configuration through a unit’s local interface, a network configuration tool or other means of configuring outside the CIP interface.

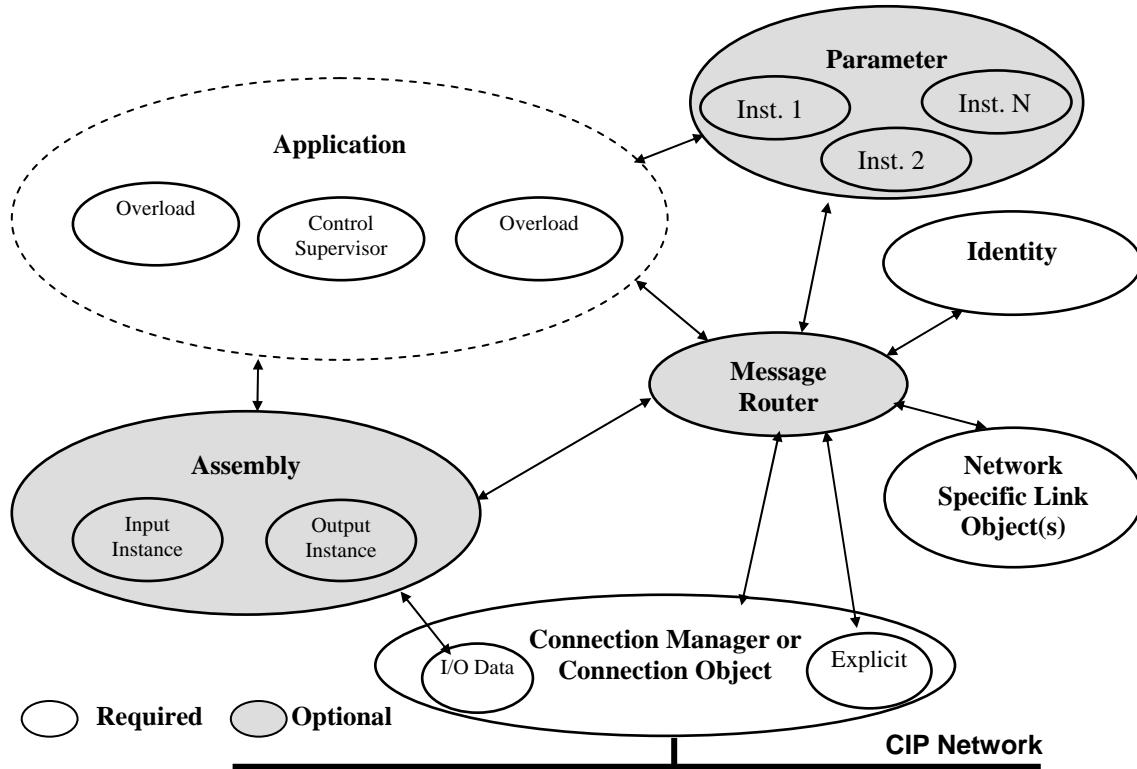
### 6-28.1 Object Model

The Object Model in Figure 6-28.2 represents a Motor Starter. The table below indicates:

- the object classes present in this device
- whether or not the class is required
- the number of instances present in each class

**Figure 6-28.1 Objects Present in a Motor Starter Device**

Object Class	Optional/Required	# of Instances
Message Router	Optional	1
CIP Common Required Objects	Required	See Section 6-2.1
Assembly	Optional	1
Parameter	Optional	-
Control Supervisor	Required	1
Overload	Required	-

**Motor Starter Device, Type: 16<sub>Hex</sub>****Figure 6-28.2 Object Model for Motor Starter Device****6-28.2 How Objects Affect Behavior**

The objects for this device affect the device's behavior as shown in the table below.

**Figure 6-28.3 Object Effect On Behavior**

Object	Effect on behavior
CIP Common Required	See Section 6-2.2 for details.
Assembly	Defines I/O data format
Parameter	Provides a public interface to device configuration data
Control Supervisor	Manages motor functions and operational states
Overload	Implements overload

**6-28.3 Defining Object Interfaces**

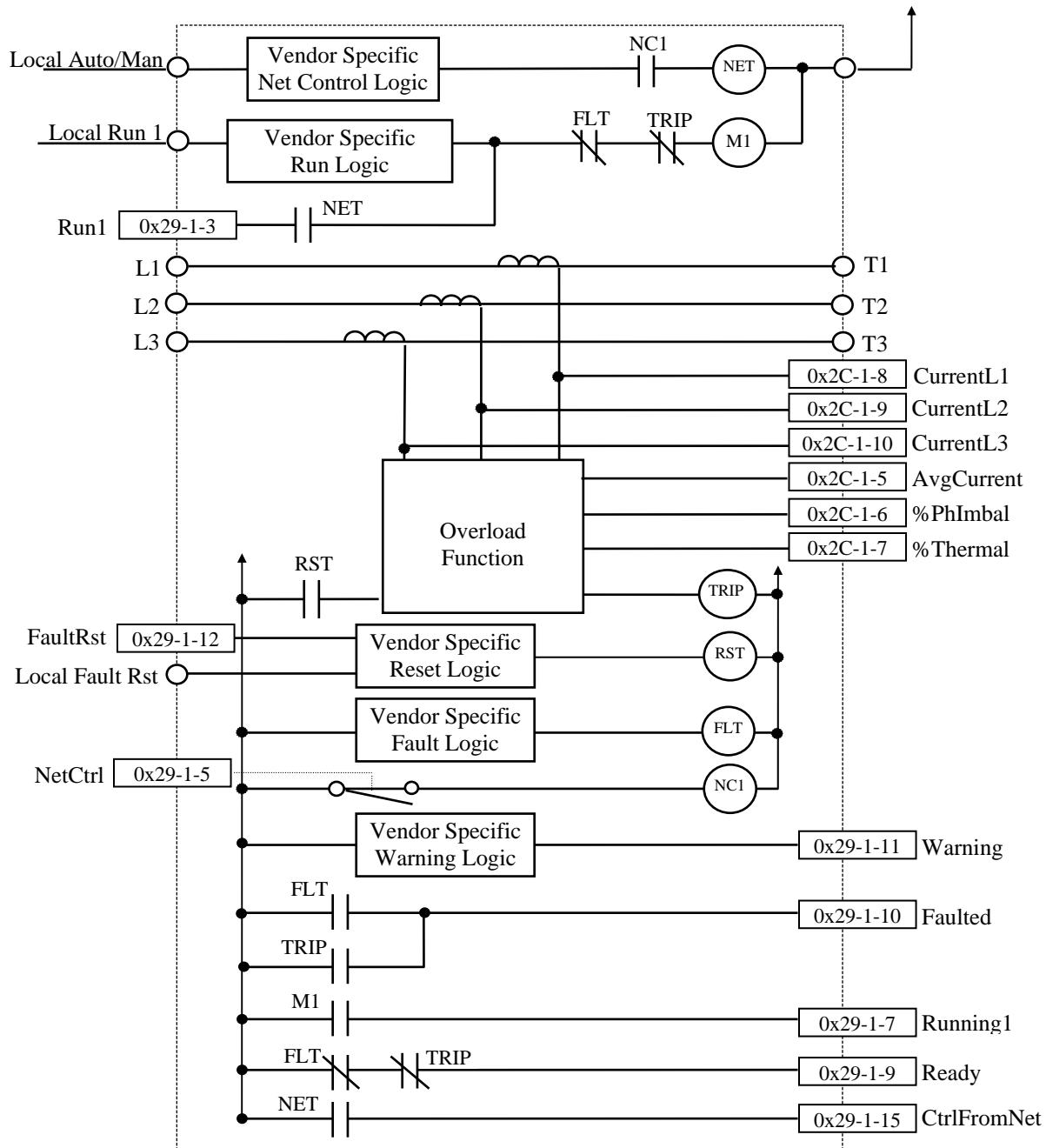
The objects in the Motor Overload have the interfaces listed in the following table:

**Figure 6-28.4 Object Interfaces**

Object	Interface
CIP Common Required	See Section 6-2.3 for details.
Assembly	I/O Connection or Message Router
Parameter	Message Router
Control Supervisor	Message Router, Assembly or Parameter Object
Overload	Message Router, Assembly or Parameter Object

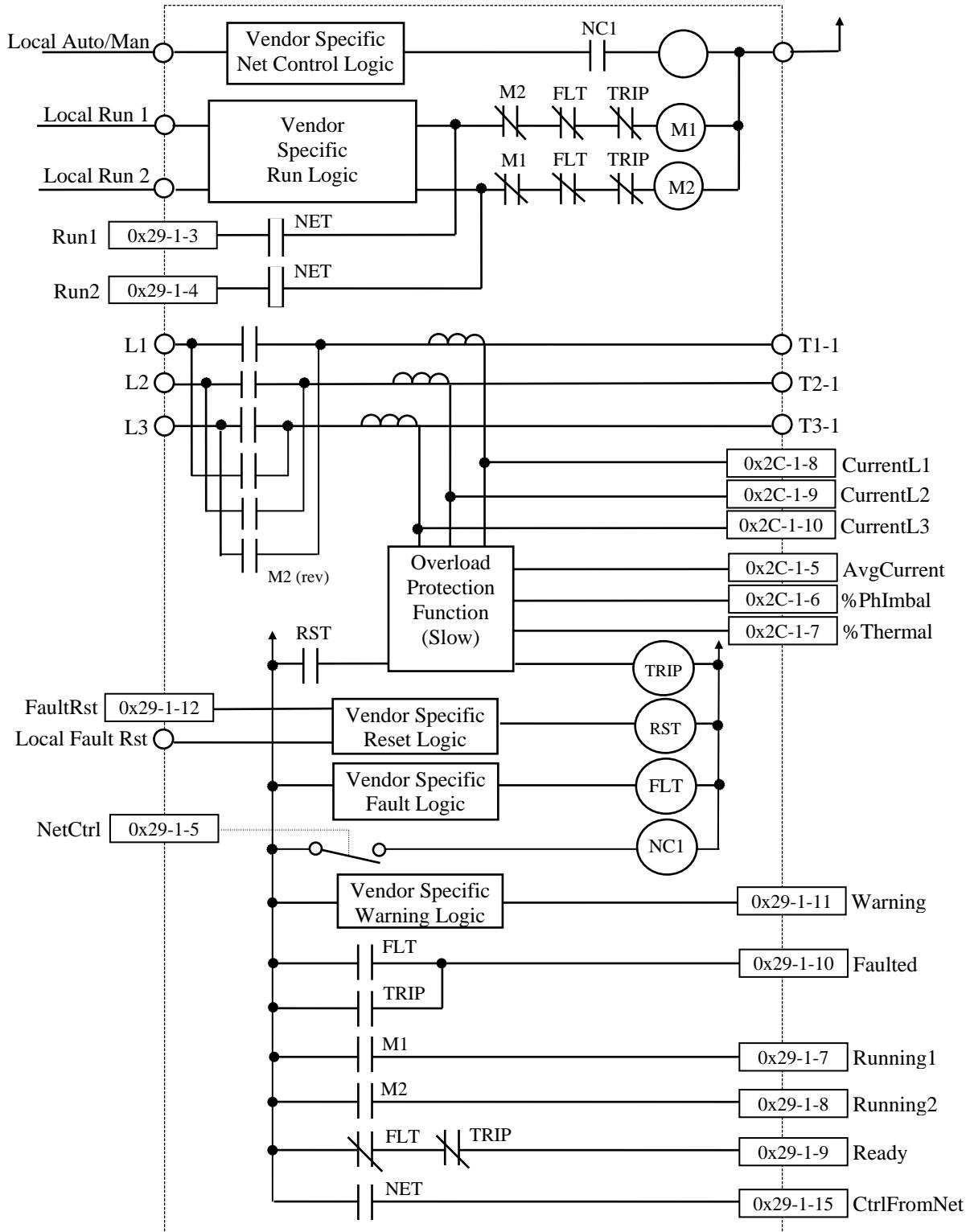
## 6-28.3.1 Starter Interface and Behavior

Figure 6-28.5 Starter Interface &amp; Behavior Diagram



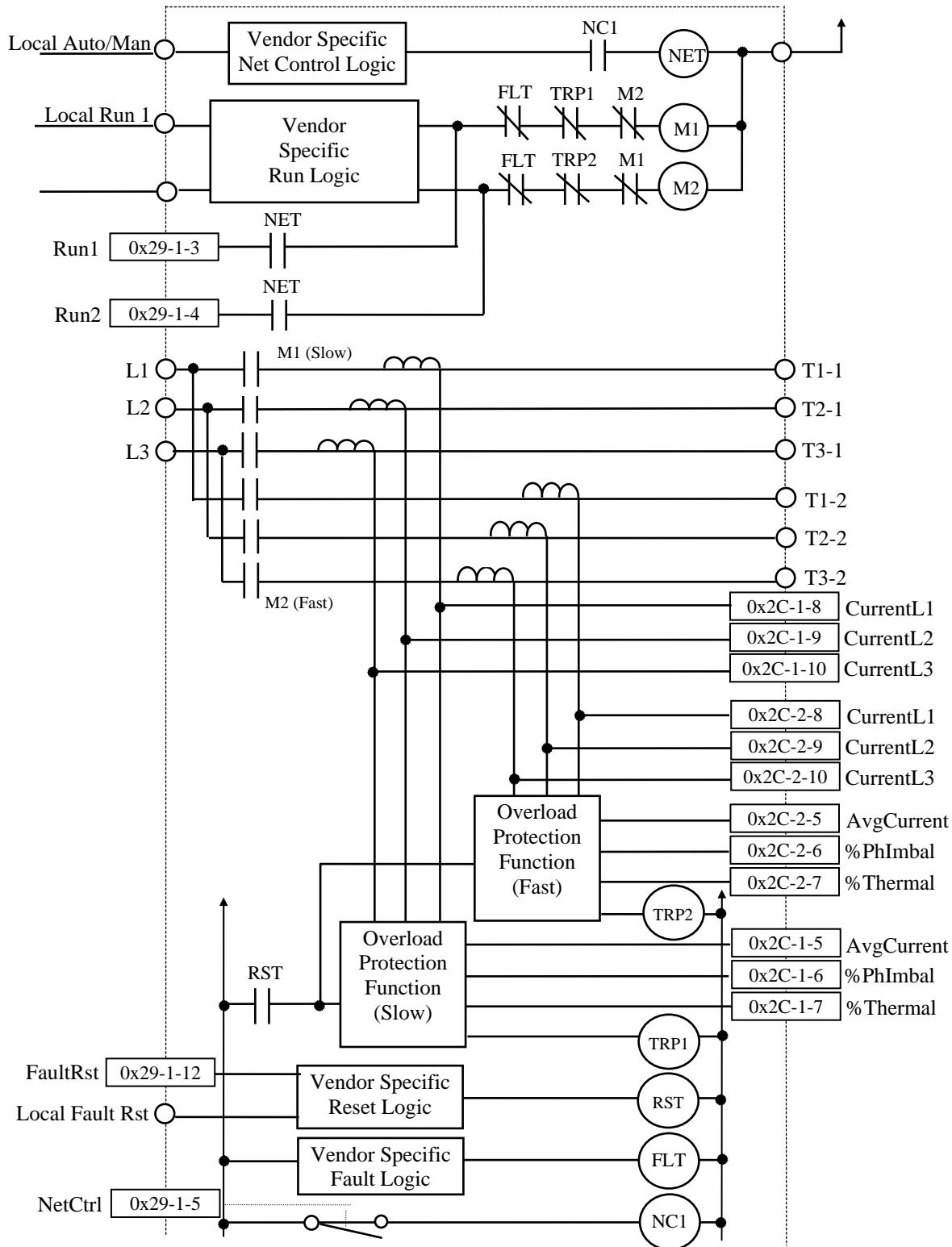
## 6-28.3.2 Reversing Motor Starter Interface and Behavior

Figure 6-28.6 Reversing Motor Starter Interface &amp; Behavior Diagram

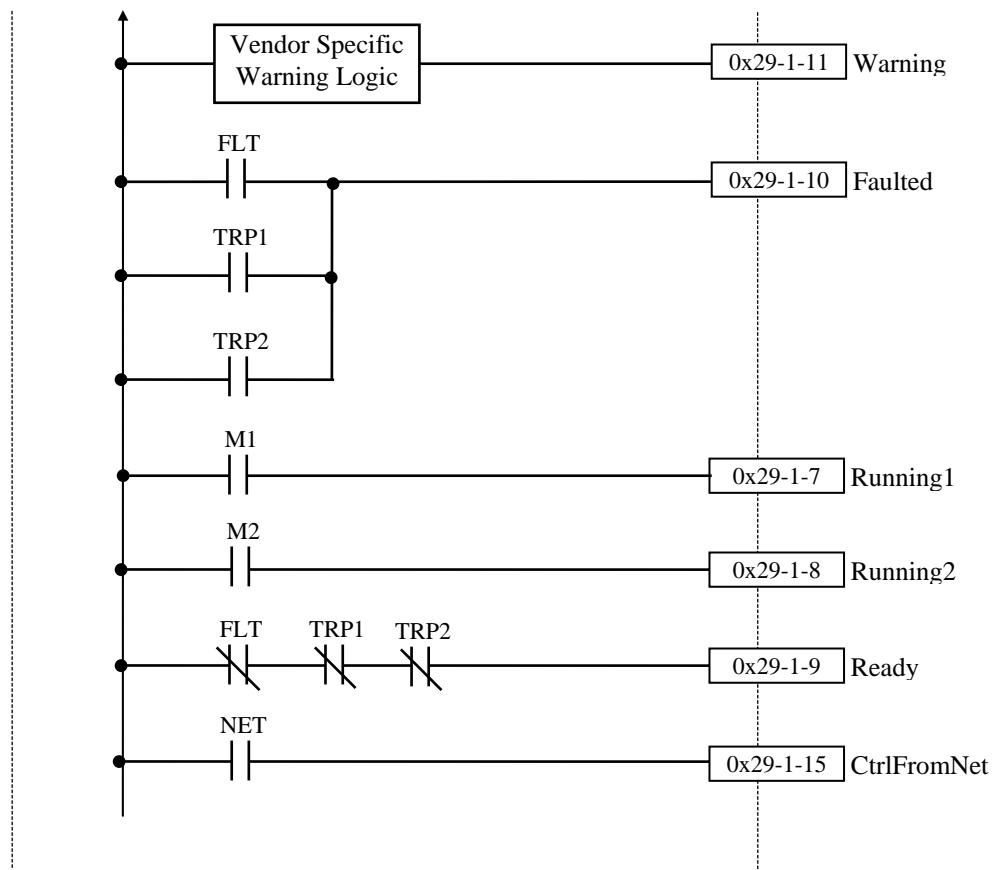


## 6-28.3.3 Two Speed Motor Starter Interface and Behavior

Figure 6-28.7 Two Speed Motor Starter Interface &amp; Behavior Diagram



**Motor Starter Device, Type: 16<sub>Hex</sub>**



## 6-28.4 I/O Assembly Instances

The IO Assembly Instance definitions in this section define the format of the “data” attribute (attribute 3) for IO Assembly Instances. Through the use of predefined instance definitions, IO Assemblies support a hierarchy of motor control devices. The device hierarchy includes motor starters, soft starters, AC and DC drives, and servo drives. Assembly Instances are numbered within the hierarchy so that each device type is assigned a range of Assembly Instance numbers, with higher functionality devices supporting higher instance numbers. **Devices in the hierarchy can choose to support instance numbers that are lower than theirs in the hierarchy.** For example a Softstart may choose to support some IO Assemblies that are defined for Overload. The following table shows the Assembly Instance numbering for the motor control device hierarchy.

**Table 6-28.8 Assembly Instance Numbering for Motor Control Device Hierarchy**

Profile	I/O Type	Instance Range
Contactors, Overloads and Starters	Output	1-19
	Input	50-69
AC/DC Drive	Output	20-29
	Input	70-79
Servo Drive	Output	30-49
	Input	80-99

The following IO Assembly Instances are defined for Motor Starters.

**Table 6-28.9 Assembly Instances for Motor Starters**

Instance	Type	Name
3	Output	Basic Motor Starter
4	Output	Extended Contactor
5	Output	Extended Motor Starter
52	Input	Basic Motor Starter
53	Input	Extended Motor Starter 1
54	Input	Extended Motor Starter 2

If a bit is not used in an IO Assembly, it is reserved for use in other Assemblies. Reserved bits in Output Assemblies are ignored by the consuming device. Reserved bits in Input Assemblies are set to zero by the producing device.

### 6-28.4.1 Connection Paths to I/O Assembly Instances

The IO Assembly Instances are chosen for IO Connections by setting the “produced\_connection\_path” (attribute 14) and “consumed\_connection\_path” (attribute 16) attributes in the appropriate connection object.

Motor Control Devices use the Symbolic Segment Type (see Appendix C) to specify paths to the IO Assembly Instances in the Motor Control Hierarchy. IO Assembly Instances are represented by ASCII strings that contain the hex number of the Assembly Instance whose path is to be chosen.

The following example shows the Symbolic Segment used to specify Output Assembly Instance 20 (14 hex).

**Motor Starter Device, Type: 16<sub>Hex</sub>****Figure 6-28.10 Symbolic Segment Example**

Segment Type			First Character					Second Character			
0	1	1	0	0	0	1	0	0	0	1	1
Segment type (symbolic)			Symbol size in bytes (2 bytes)					ASCII 1 (31 hex)			
								ASCII 4 (34 hex)			

**6-28.5 I/O Assembly Data Attribute Format****6-28.5.1 Output Assembly Data Attribute Format****Table 6-28.11 Instance 3: Basic Motor Starter**

This is the only required output assembly for device type Motor Starter (16hex)								
Byte	Bit 7	Bit 6	Bit .5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Reserved	Reserved	Reserved	Reserved	Reserved	FaultReset	Reserved	Run1

**Table 6-28.12 Instance 4: Extended Contactor (see table for functional assignments)**

This assembly uses some optional attributes.								
Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Run2	Run1

**Table 6-28.13 Instance 5: Extended Motor Starter (see table for functional assignments)**

This assembly uses some optional attributes..								
Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Reserved	Reserved	Reserved	Reserved	Reserved	FaultReset	Run2	Run1

**6-28.5.2 Input Assembly Data Attribute Format****Table 6-28.14 Instance 52: Basic Motor Starter**

This is the only required input assembly for Motor Starter (16hex)								
Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Reserved	Reserved	Reserved	Reserved	Reserved	Running1	Reserved	Faulted/ Trip

**Table 6-28.15 Instance 53: Extended Motor Starter 1 (see table for functional assignments)**

This assembly uses some optional attributes								
Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Reserved	Reserved	Cntrlfrom Net	Ready	Reserved	Running1	Warning	Faulted/ Trip

**Table 6-28.16 Instance 54: Extended Motor Starter 2 (see table for functional assignments)**

This assembly uses some optional attributes								
Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Reserved	Reserved	Cntrlfrom Net	Ready	Running2	Running1	Warning	Faulted/ Trip

## 6-28.6 Mapping I/O Assembly Data Attribute Components

### 6-28.6.1 Mapping for Motor Starter Output Assembly Data Components

Table 6-28.17 Output Data Assembly Mapping

Data Name	Class Name	Class Number	Instance	Attribute Name	Attribute Number
Run1	Control Supervisor	0x29	1	Run1	3
Run2	Control Supervisor	0x29	1	Run2	4
Fault Reset	Control Supervisor	0x29	1	FaultRst	12

### 6-28.6.2 Mapping for Motor Starter Input Assembly Data Components

Table 6-28.18 Input Data Assembly Mapping

Data Name	Class Name	Class Number	Instance	Attribute Name	Attribute Number
Faulted/Trip	Control Supervisor	0x29	1	Faulted	10
Warning	Control Supervisor	0x29	1	Warning	11
Running1	Control Supervisor	0x29	1	Running1	7
Running2	Control Supervisor	0x29	1	Running2	8
Ready	Control Supervisor	0x29	1	Ready	9
Control From Net	Control Supervisor	0x29	1	CtrlFromNet	15

## 6-28.7 Defining Device Configuration

Public access to the Control Supervisor Object and the Overload Object must be supported for configuration of Motor Starter devices. If supported, optional Parameter Objects may be used to access the various configuration attributes in the Control Supervisor Object and the Overload Object.

## 6-29 Softstart Starter Device

### Device Type: 17 hex

The Softstart Starter device profile is part of a “Hierarchy of Motor Control Devices” that are supported by CIP. This hierarchy includes:

- Contactors, Overloads, and Across the Line Motor Starters
- Softstarters
- AC/DC Drives
- Servo Drives

Devices within this hierarchy use a common Control Supervisor object to control state behavior of the device. Devices within this hierarchy also support a hierarchy of “IO Assembly Instance” definitions which are used to pass control and status information to and from a device. Assembly instances are numbered so that each device type is assigned a range of instance numbers, with higher functionality devices supporting higher instance numbers. Devices within the hierarchy can choose to support some instance numbers that are lower than theirs in the hierarchy. For example, an AC Drive may choose to support some instances that are defined for Across the Line Motor Starters. This makes it easier to interchange drives and starters within a system.

This profile makes Softstart Starters of the same device type inter-operable, but not directly interchangeable without doing configuration through a unit’s local interface, a network configuration tool or other means of configuring outside the CIP interface.

### 6-29.1 Object Model

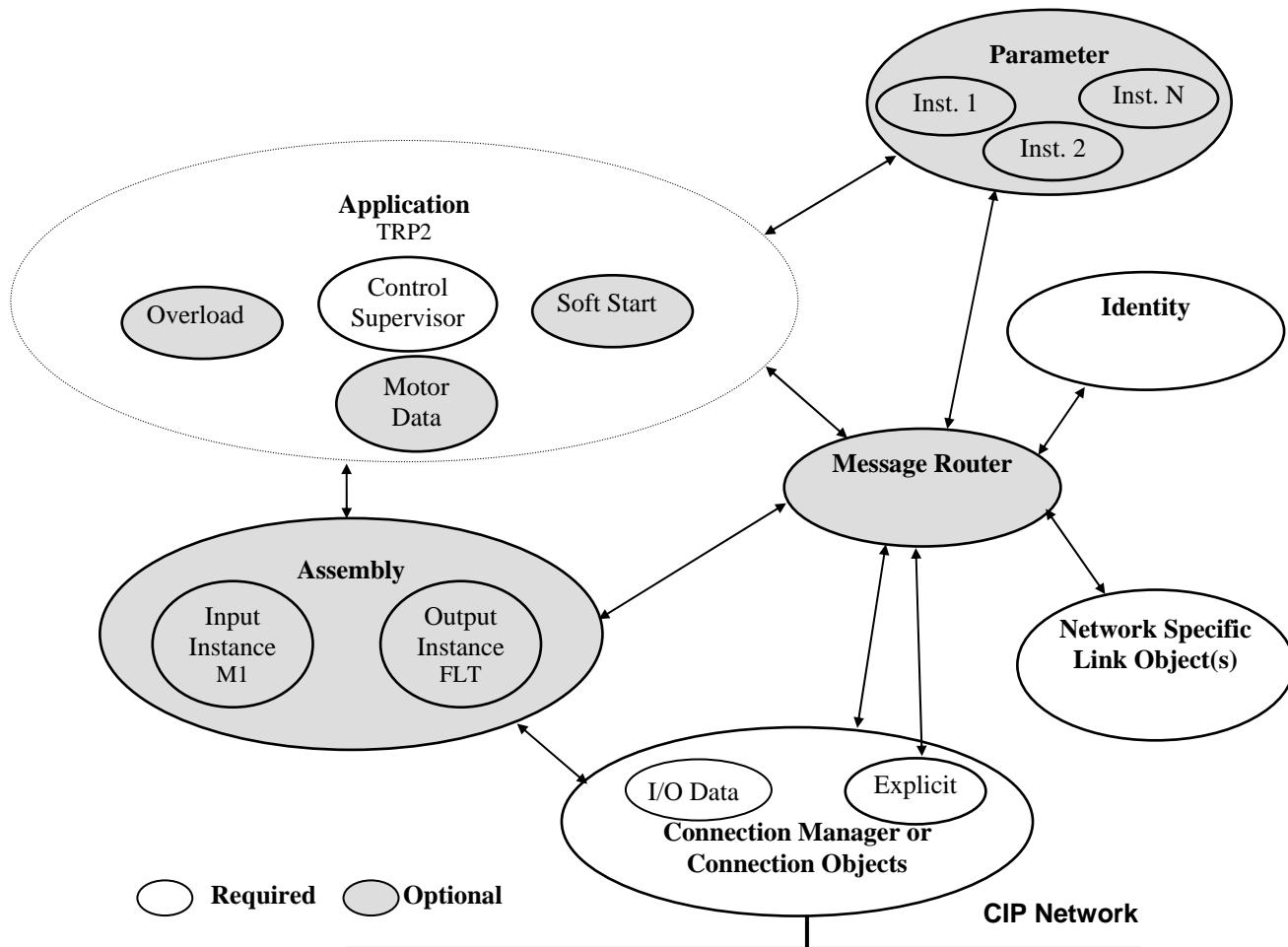
The Object Model in Figure 6-29.2 represents a Softstart Starter. The table below indicates:

- the object classes present in this device
- whether or not the class is required
- the number of instances present in each class

**Table 6-29.1 Object Classes Present in a Softstart Starter Device**

Object Class	Optional/Required	# of Instances
Message Router	Optional	1
CIP Common Required Objects	Required	See Section 6-2.1
Assembly	Optional	1
Parameter	Optional	-
Control Supervisor	Required	1
Softstart	Optional	-
Overload	Optional	-
Motor Data	Optional	-

Figure 6-29.2 Object Model for Softstart Starter Device



## 6-29.2 How Objects Affect Behavior

The objects for this device affect the device's behavior as shown in the table below.

Table 6-29.3 Object Effect on Behavior

Object	Effect on behavior
CIP Common Required	See Section 6-2.2 for details.
Assembly	Defines I/O data format
Parameter	Provides a public interface to device configuration data
Control Supervisor	Manages motor functions and operational states
Softstart	Implements the Softstart functions
Overload	Implements overload
Motor Data	Define motor data for motor connected to this device.

### 6-29.3 Defining Object Interfaces

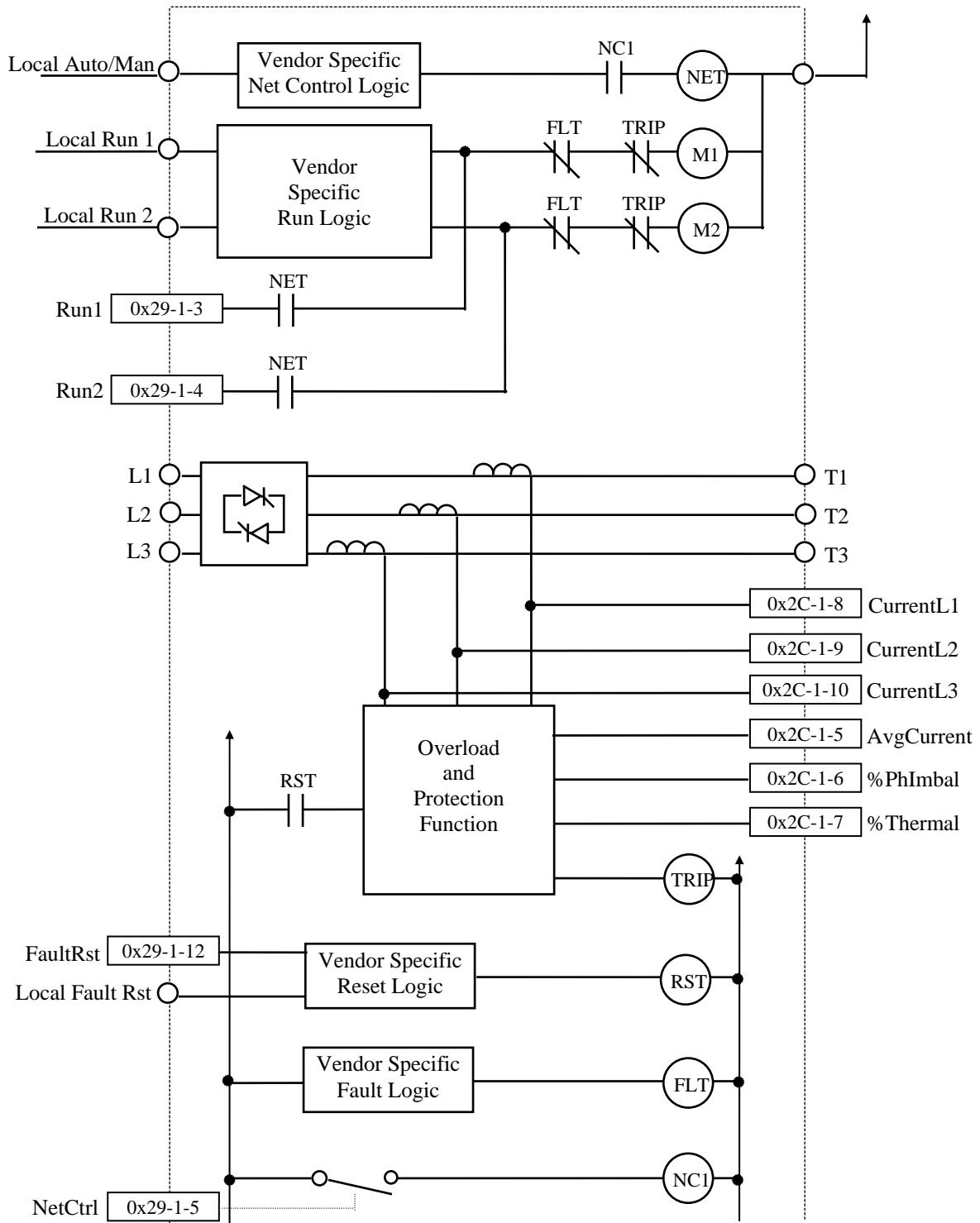
The objects in the Motor Overload have the interfaces listed in the following table:

**Table 6-29.4 Object Interfaces**

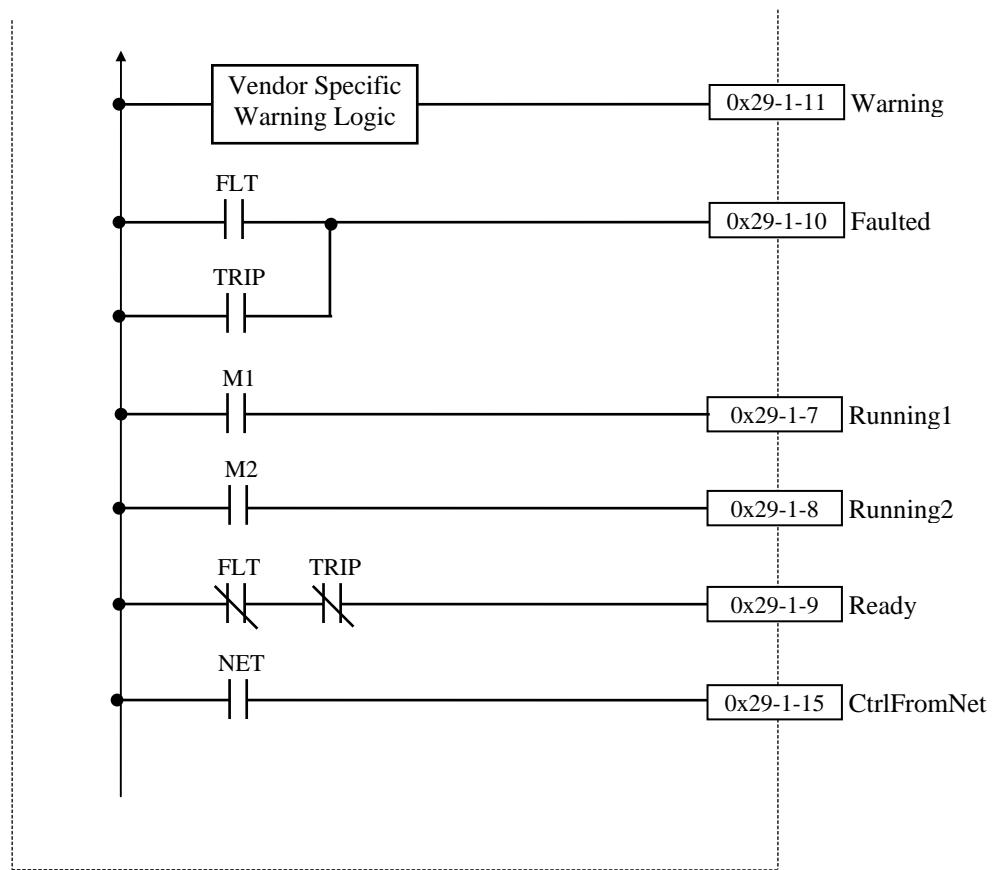
Object	Interface
CIP Common Required	See Section 6-2.3 for details.
Assembly	I/O Connection or Message Router
Parameter	Message Router
Control Supervisor	Message Router, Assembly or Parameter Object
Softstart	Message Router or Assembly
Overload	Message Router, Assembly or Parameter Object
Motor Data	Message Router, Parameter Object

## 6-29.4 Softstart Motor Interface and Behavior

Figure 6-29.5 Softstart Motor Interface &amp; Behavior Diagram



Softstart Starter Device, Type: 17<sub>Hex</sub>



## 6-29.5 I/O Assembly Instances

The IO Assembly Instance definitions in this section define the format of the “data” attribute (attribute 3) for IO Assembly Instances. Through the use of predefined instance definitions, IO Assemblies support a hierarchy of motor control devices. The device hierarchy includes motor starters, soft starters, AC and DC drives, and servo drives. Assembly Instances are numbered within the hierarchy so that each device type is assigned a range of Assembly Instance numbers, with higher functionality devices supporting higher instance numbers. **Devices in the hierarchy can choose to support instance numbers that are lower than theirs in the hierarchy.** For example a Softstart may choose to support some IO Assemblies that are defined for Overload. The following table shows the Assembly Instance numbering for the motor control device hierarchy.

**Table 6-29.6 I/O Assembly Instance Numbering Ranges**

Profile	I/O Type	Instance Range
Contactors, Overloads and Starters	Output	1-19
	Input	50-69
AC/DC Drive	Output	20-29
	Input	70-79
Servo Drive	Output	30-49
	Input	80-99

The following IO Assembly Instances are defined for Softstarters.

**Table 6-29.7 I/O Assembly Instances for Softstarters**

Instance	Type	Name
60	Input	Basic Softstart
61	Input	Extended Softstart

### 6-29.5.1 Connection Paths to I/O Assembly Instances

The IO Assembly Instances are chosen for IO Connections by setting the “produced\_connection\_path” (attribute 14) and “consumed\_connection\_path” (attribute 16) attributes in the appropriate connection object.

Motor Control Devices use the Symbolic Segment Type (see Appendix C) to specify paths to the IO Assembly Instances in the Motor Control Hierarchy. IO Assembly Instances are represented by ASCII strings that contain the hex number of the Assembly Instance whose path is to be chosen.

The following example shows the Symbolic Segment used to specify Output Assembly Instance 20 (14 hex).

**Table 6-29.8 Symbolic Segment Example**

Segment Type	First Character	Second Character
0 1 1   0 0 0 1 0 Segment type   Symbol size in bytes (2 bytes)	0 0 1 1 0 0 0 1 ASCII 1 (31 hex)	0 0 1 1 0 1 0 0 ASCII 4 (34 hex)

## 6-29.6 I/O Assembly Data Attribute Format

### 6-29.6.1 Output Assembly Data Attribute Format

There are no new output assemblies defined for Softstart devices.

### 6-29.6.2 Input Assembly Data Attribute Format

**Table 6-29.9 Instance 60: Basic Softstart Input**

This is the only required input assembly, for the device type SoftStart (15hex)								
Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	At Reference	Reserved	Reserved	Reserved	Reserved	Running1	Reserved	Faulted/Trip

**Table 6-29.10 Instance 61: Extended Softstart Input (see table for functional assignments)**

This assembly uses some of the optional attributes.								
Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	At Reference	Reserved	CntrlFromNet	Ready	Running2	Running1	Warning	Faulted/Trip

## 6-29.7 Mapping I/O Assembly Data Attribute Components

**Table 6-29.11 Assembly Data Attribute Mapping**

Data Name	Class Name	Class Number	Instance	Attribute Name	Attribute Number
Faulted/ Trip	Control Supervisor	0x29	1	Faulted/ Trip	10
Running_1	Control Supervisor	0x29	1	Running_1	7
Running_2	Control Supervisor	0x29	1	Running_2	8
Ready	Control Supervisor	0x29	1	Ready	9
Warning	Control Supervisor	0x29	1	Warning	11
Control From Net	Control Supervisor	0x29	1	CtrlFromNet	15
At Reference	Softstart	0x2D	1	AtRef	1

## 6-29.8 Defining Device Configuration

Public access to the Control Supervisor Object and the Overload Object must be supported for configuration of Softstart devices. If supported, optional Parameter Objects may be used to access the various configuration attributes in the Control Supervisor Object and the Overload Object.

## 6-30 Human-Machine Interface (HMI)

### Device Type: 18<sub>hex</sub>

The Human-Machine Interface (HMI) Device type is based on the Generic Device Type (00hex). The purpose of this device profile is to allow network tools to identify and distinguish HMI devices from other Generic devices on a network. Over time, the and ControlNet International and Open DeviceNet Vendor Association's HMI Special Interest Groups (SIG) will enhance the HMI device profile to define minimum required objects and optional objects which are similar among HMI devices. HMI Device type devices are not interchangeable.

### 6-30.1 Object Model

The Object Model in Figure 6-30.2 represents the minimum support in an HMI Device. The table below indicates:

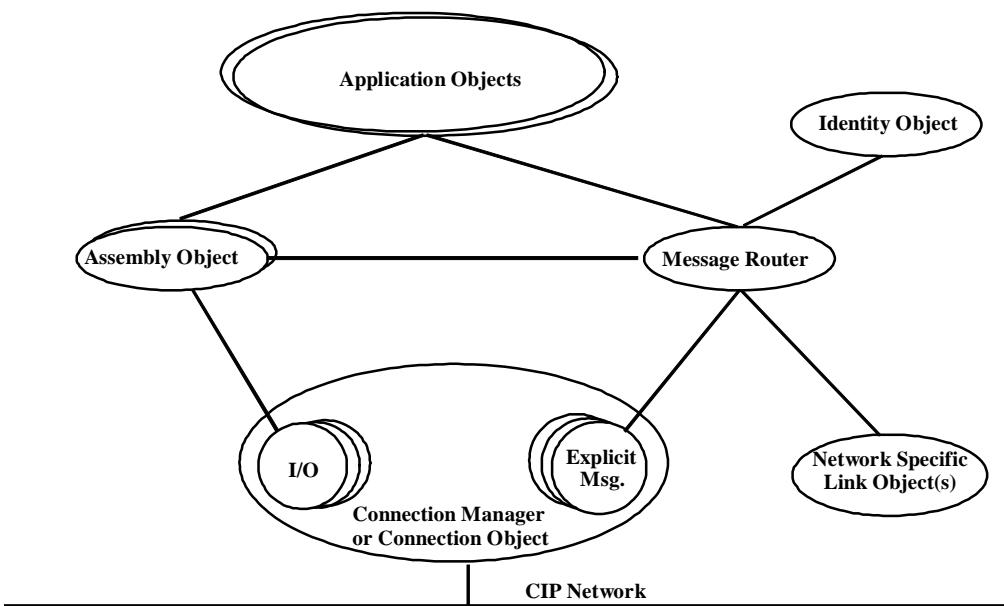
- the object classes present in this device
- whether or not the class is required
- the number of instances present in each class

**Table 6-30.1 Objects Present in an HMI Device**

Object Class	Optional/Required	# of Instances
CIP Common Required Objects	Required	See Section 6-2.1
Assembly	Required	at least 1
Application	Required	at least 1

The HMI Device profile cannot specify the definition of the Assembly Object or the type of application objects necessary for device operation. This portion of the device profile must be supplied by the product developer as described in Section 6-1 through 6-7 inclusive.

**Figure 6-30.2 Object Model for an HMI Device**



## 6-30.2 How Objects Affect Behavior

The objects for this device affect the device's behavior as shown in the table below.

**Table 6-30.3 Object Effect on Behavior**

Object	Effect on behavior
CIP Common Required	See Section 6-2.2 for details.
Assembly	Defines input/output and configuration data format
Application	Defines device operation

## 6-30.3 Defining Object Interfaces

The objects in the HMI Device have the interfaces listed in the following table:

**Table 6-30.4 Object Interfaces**

Object	Interface
CIP Common Required	See Section 6-2.3 for details.
Assembly	I/O Connection or Message Router
Application	Assembly or Message Router

## 6-31 Mass Flow Controller Device

### Device Type: 1A<sub>hex</sub>

A Mass Flow Controller is a device that measures and controls the mass flow rate of gas or liquid. It contains three principle components: a mass flow rate sensor which can be one of a variety of types, including thermal or pressure-based; a mass flow rate metering valve which can be actuated by one of a variety of actuator types, including solenoid, voice coil or piezo; and, a controller which closes the loop by receiving a setpoint and driving the actuator such that the mass flow rate is controlled to the setpoint.

### 6-31.1 Object Model

The Object Model in Figure 6-31.4 represents a Mass Flow Controller Device. The table below indicates:

- the object classes present in this device
- whether or not the class is required
- the number of instances present in each class

**Table 6-31.1 Objects Present in a Mass Flow Controller Device**

Object Class	Optional/Required	# of Instances
CIP Common Required Objects	Required	See Section 6-2.1
Assembly	Required	at least 1 Input and 1 Output
S-Device Supervisor	Required	1
S-Gas Calibration	Optional	0 or More
S-Analog Sensor	Required	1
S-Analog Actuator	Conditional *	1
S-Single Stage Controller	Conditional *	1

\* Required for a Mass Flow Controller, a device that contains a Valve and a Controller.  
Not supported in a Mass Flow Meter Device (an MFC without a Valve or a Controller).

#### 6-31.1.1 Class Subclasses

Each class level subclass defines a unique meaning for an overlapping range of class attribute IDs and/or class service IDs. The range for subclass definitions begins at ID 96 and numbers downward for attributes, and ID 63<sub>hex</sub> and numbers downward for services. The subclass for a given class is identified by the value of its Subclass class attribute. There are no class level subclasses specified for this device.

#### 6-31.1.2 Instance Subclasses

Each instance level subclass defines a unique meaning for an overlapping range of instance attribute IDs and/or instance service IDs. The range for subclass definitions begins at ID 96 and numbers downward for attributes, and ID 63<sub>hex</sub> and numbers downward for services. The subclass for a given instance is identified by the value of its Subclass instance attribute. The following tables identify which object instance IDs are assigned subclasses for this device.

**Table 6-31.2 S-Analog Sensor Object Subclasses**

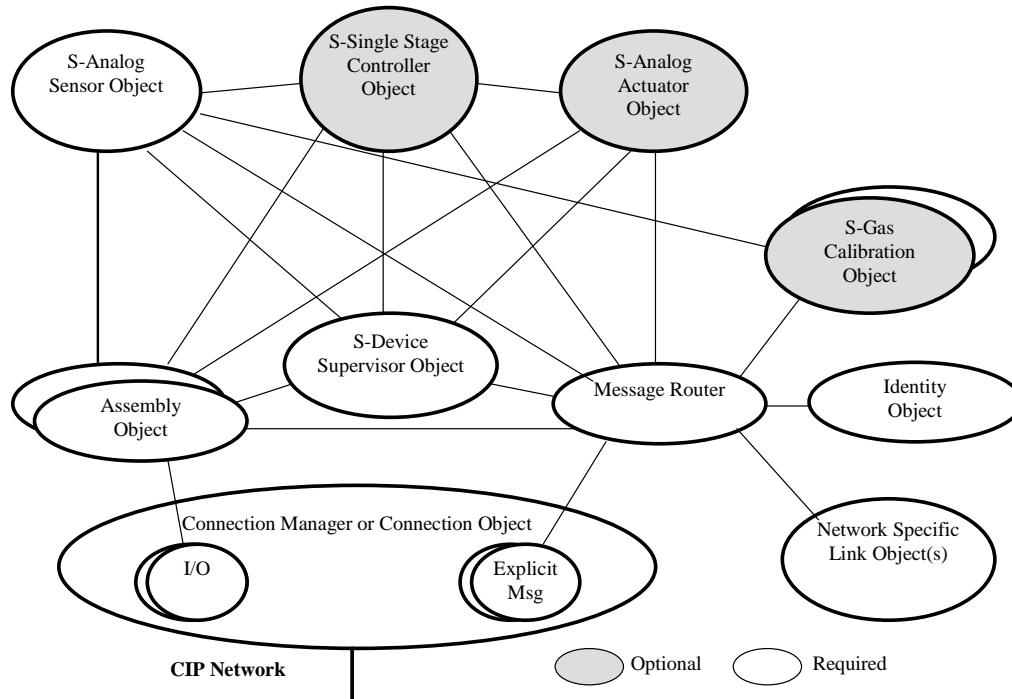
Instance ID	Subclass Name	Subclass ID (Attribute 99 Value)	Required	Function	Restrictions
1	Flow Diagnostics	01	Required	Added diagnostics for MFC	None

Mass Flow Controller Device, Type: 1A<sub>Hex</sub>

Table 6-31.3 S-Gas Calibration Object Subclasses

Instance ID	Subclass Name	Subclass ID (Attribute 99 Value)	Required	Function	Restrictions
1	Standard T & P	01	Optional	Standard Temperature and Pressure	None

Figure 6-31.4 Object Model for the MFC Device



## 6-31.2 How Objects Affect Behavior

Table 6-31.5 Object Affect on Behavior

Object	Effect on behavior
CIP Common Required	See Section 6-2.2 for details.
Assembly	Defines input/output and configuration data format
S-Device Supervisor	<p>Supports the Stop, Start, Reset, Abort, Recover and Perform_Diagnostic services for ALL Application Objects in the device and consolidates the Exception Conditions and Application Objects' Status.</p> <p>This object behaves differently from the Identity Object in that the S-Device Supervisor object provides a single point of access to the Application Objects only; it does not effect the CIP specific objects (i.e., Identity, Network Specific Link Object, Connection, etc.).</p>
S-Gas Calibration	Modifies the correction algorithm of the S-Analog Sensor object which includes the selection mechanism to enable an S-Gas Calibration object instance.
S-Analog Sensor	Feeds the process variable to the Single Stage Controller object
S-Single Stage Controller	Feeds the control variable to the Analog Actuator object
S-Analog Actuator	Operates the Flow Control Valve of the device

### **6-31.3 Defining Object Interfaces**

**Table 6-31.6 Object Interfaces**

<b>Object</b>	<b>Interface</b>
CIP Common Required	See Section 6-2.3 for details.
Assembly	I/O Connection or Message Router
S-Device Supervisor	Assembly or Message Router
S-Gas Calibration	Message Router
S-Analog Sensor	Assembly or Message Router
S-Single Stage Controller	Assembly or Message Router
S-Analog Actuator	Assembly or Message Router

## 6-31.4 I/O Assembly Instances

The following table identifies the I/O assembly instances supported by the MFC.

**Table 6-31.7 I/O Assembly Instances**

Number	Required	Type	Name
1	N	Input	Flow
2	Y (default)	Input	Status and Flow
3	N	Input	Status, Flow and Valve
4	N	Input	Status, Flow, and Setpoint
5	N	Input	Status, Flow, Setpoint and Valve
6	Y	Input	Status, Flow, Setpoint, Override and Valve
7	Y (default)	Output	Setpoint
8	Y	Output	Override and Setpoint
9	N	Input	Status
10	N	Input	Exception Detail Alarm
11	N	Input	Exception Detail Warning
12	N	Input	Exception Detail Alarm and Exception Detail Warning
13	N	Input	FP-Flow
14	Y	Input	FP-Status and Flow
15	N	Input	FP-Status, Flow and Valve
16	N	Input	FP-Status, Flow, and Setpoint
17	N	Input	FP-Status, Flow, Setpoint and Valve
18	Y	Input	FP-Status, Flow, Setpoint, Override and Valve
19	Y	Output	FP-Setpoint
20	Y	Output	FP-Override and Setpoint

## 6-31.5 I/O Assembly Object Instance Data Attribute Format

The manufacturer of a Mass Flow Controller Device must specify which Assembly instances are supported by the device.

The I/O Assembly DATA attribute has the format shown below.

**Table 6-31.8 I/O Assembly Data Attribute Format**

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	0	Flow (low byte)							
	1	Flow (high byte)							
2	0	Status							
	1	Flow (low byte)							
	2	Flow (high byte)							
3	0	Status							
	1	Flow (low byte)							
	2	Flow (high byte)							
	3	Valve (low byte)							
	4	Valve (high byte)							
4	0	Status							
	1	Flow (low byte)							
	2	Flow (high byte)							
	3	Setpoint (low byte)							
	4	Setpoint (high byte)							
5	0	Status							
	1	Flow (low byte)							
	2	Flow (high byte)							
	3	Setpoint (low byte)							
	4	Setpoint (high byte)							
	5	Valve (low byte)							
	6	Valve (high byte)							
6	0	Status							
	1	Flow (low byte)							
	2	Flow (high byte)							
	3	Setpoint (low byte)							
	4	Setpoint (high byte)							
	5	Override							
	6	Valve (low byte)							
	7	Valve (high byte)							
7	0	Setpoint (low byte)							
	1	Setpoint (high byte)							
	0	Override							
8	1	Setpoint (low byte)							
	2	Setpoint (high byte)							

Mass Flow Controller Device, Type: 1A<sub>Hex</sub>

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
9	0	Status							
10	0	Status							
	1	Exception Detail Alarm 0 (size, common)							
	2	Exception Detail Alarm 1 (common 0)							
	3	Exception Detail Alarm 2 (common 1)							
	4	Exception Detail Alarm 3 (size, device)							
	5	Exception Detail Alarm 4 (device 0)							
	6	Exception Detail Alarm 5 (size, manufacturer)							
	7	Exception Detail Alarm 6 (manufacturer 0)							
11	0	Status							
	1	Exception Detail Warning 0 (size, common)							
	2	Exception Detail Warning 1 (common 0)							
	3	Exception Detail Warning 2 (common 1)							
	4	Exception Detail Warning 3 (size, device)							
	5	Exception Detail Warning 4 (device 0)							
	6	Exception Detail Warning 5 (size, manufacturer)							
	7	Exception Detail Warning 6 (manufacturer, 0)							
12	0	Status							
	1	Exception Detail Alarm 0 (size, common)							
	2	Exception Detail Alarm 1 (common 0)							
	3	Exception Detail Alarm 2 (common 1)							
	4	Exception Detail Alarm 3 (size, device)							
	5	Exception Detail Alarm 4 (device 0)							
	6	Exception Detail Alarm 5 (size, manufacturer)							
	7	Exception Detail Alarm 6 (manufacturer, 0)							
	8	Exception Detail Warning 0 (size, common)							
	9	Exception Detail Warning 1 (common 0)							
	10	Exception Detail Warning 2 (common 1)							
	11	Exception Detail Warning 3 (size, device)							
	12	Exception Detail Warning 4 (device 0)							
	13	Exception Detail Warning 5 (size, manufacturer)							
	14	Exception Detail Warning 6 (manufacturer, 0)							
13	0	Flow (low byte)							
	1	Flow							
	2	Flow							
	3	Flow (high byte)							

Mass Flow Controller Device, Type: 1A<sub>Hex</sub>

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
14	0	Status							
	1	Flow (low byte)							
	2	Flow							
	3	Flow							
	4	Flow (high byte)							
15	0	Status							
	1	Flow (low byte)							
	2	Flow							
	3	Flow							
	4	Flow (high byte)							
	5	Valve (low byte)							
	6	Valve							
	7	Valve							
	8	Valve (high byte)							
16	0	Status							
	1	Flow (low byte)							
	2	Flow							
	3	Flow							
	4	Flow (high byte)							
	5	Setpoint (low byte)							
	6	Setpoint							
	7	Setpoint							
	8	Setpoint (high byte)							
	9	Setpoint (low byte)							
17	10	Setpoint							
	11	Setpoint							
	12	Setpoint (high byte)							
	13	Valve (low byte)							
	14	Valve							
	15	Valve							
	16	Valve (high byte)							

Mass Flow Controller Device, Type: 1A<sub>Hex</sub>

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
18	0	Status							
	1	Flow (low byte)							
	2	Flow							
	3	Flow							
	4	Flow (high byte)							
	5	Setpoint (low byte)							
	6	Setpoint							
	7	Setpoint							
	8	Setpoint (high byte)							
	9	Override							
	10	Valve (low byte)							
	11	Valve							
	12	Valve							
	13	Valve (high byte)							
19	0	Setpoint (low byte)							
	1	Setpoint							
	2	Setpoint							
	3	Setpoint (high byte)							
20	0	Override							
	1	Setpoint (low byte)							
	2	Setpoint							
	3	Setpoint							
	4	Setpoint (high byte)							

**6-31.6 Mapping I/O Assembly Data Attribute Components**

Each of the *S-Analog Sensor*, *S-Analog Actuator* and *S-Single Stage Controller* object definitions specifies a behavior that modifies the *Data Type* of certain attributes based upon the first valid I/O connection established to an Assembly Object instance. In order to maintain consistency, this device type will only allow connections to either INT or REAL based Assembly instances. Once a valid connection is established, attempts to configure connections to a different type of Assembly instance will return an error.

The following table indicates the I/O assembly Data attribute mapping for this MFC device.

Mass Flow Controller Device, Type: 1A<sub>Hex</sub>

Table 6-31.9 I/O Assembly Data Mapping

Data Component Name	Class		Instance Number	Attribute		
	Name	Number		Name	Number	Type
Flow	S-Analog Sensor	31 <sub>hex</sub>	1	Indicated Flow	6	INT
Valve	S-Analog Actuator	32 <sub>hex</sub>	1	Value	6	INT
Override	S-Analog Actuator	32 <sub>hex</sub>	1	Override	5	USINT
Setpoint	S-Single Stage Controller	33 <sub>hex</sub>	1	Setpoint	6	INT
Status	S-Device Supervisor	30 <sub>hex</sub>	1	Exception Status	12	BYTE
Exception Detail Alarm	S-Device Supervisor	30hex	1	Exception Detail Alarm	13	STRUCT
Exception Detail Warning	S-Device Supervisor	30hex	1	Exception Detail Warning	14	STRUCT
FP-Flow	S-Analog Sensor	31 <sub>hex</sub>	1	Indicated Flow	6	REAL
FP-Valve	S-Analog Actuator	32 <sub>hex</sub>	1	Value	6	REAL
FP-Setpoint	S-Single Stage Controller	33 <sub>hex</sub>	1	Setpoint	6	REAL

## 6-31.7 Object Limitations and Specific Definitions

This section describes limitations and specific definitions applicable to the listed objects of this device profile when these objects are applied in this type of device.

### 6-31.7.1 S-Device Supervisor Object Instance

#### 6-31.7.1.1 Limitations

Table 6-31.10 Device Type Attribute Limitations

Attribute	Limitation
Device Type	Supported Values: “MFC” = Mass Flow Controller device “MFM” = Mass Flow Meter device

#### 6-31.7.1.2 Specific Definition

The following table specifies the data attribute bit mapping for the **Device Exception Detail** bytes for this MFC device. For more descriptive information, see the definition of the S-Device Supervisor Object Class. Noted, for each entry, is the Object from which the Status byte/bit is mapped. See the object specification for the detailed bit mapping.

Any Exception Bit not supported must default to 0. Note that this profile allows for only one byte of manufacturer specific exception detail.

**Mass Flow Controller Device, Type: 1A<sub>Hex</sub>****Table 6-31.11 Device Exception Detail Attribute Mapping**

<b>Data Component</b>	<b>Bit 7</b>	<b>Bit 6</b>	<b>Bit 5</b>	<b>Bit 4</b>	<b>Bit 3</b>	<b>Bit 2</b>	<b>Bit 1</b>	<b>Bit 0</b>
MFC Device Exception Detail Size	0	0	0	0	0	0	0	1
MFC Device Exception Detail	Reserved 0	Reserved 0	Valve High S-Analog Actuator	Valve Low S-Analog Actuator	Flow Control S-Single Stage Controller	Flow High S-Analog Sensor	Flow Low S-Analog Sensor	Reading Valid * S-Analog Sensor
Manufacturer Exception Detail Size	0	0	0	0	0	0	0	1
Manufacturer Exception Detail	8 Bits defined by Manufacturer							

\* Only used in the Warning Exception Detail, this bit is always = 0 in the Alarm Exception Detail.

**6-31.7.2 S-Analog Sensor Object Instance****6-31.7.2.1 Limitations****Table 6-31.12 Attribute Limitations**

<b>Attribute</b>	<b>Limitation</b>	<b>Requirement</b>	<b>Default</b>
Data Type	Supported Values = {INT; REAL}	Supported Values = {INT; REAL}	INT
Data Units	Supported Values = {Counts & Units of the Flow Group} (see Appendix D)	Supported Values = {Counts; sccm}	Counts
Offset-A Data Type	Supported Values = {INT; REAL}	Supported Values = {INT; REAL}	INT
Gain Data Type	Supported Values = {INT; REAL}	Supported Values = {INT; REAL}	INT

**6-31.7.3 S-Analog Actuator Object Instance****6-31.7.3.1 Limitations****Table 6-31.13 Attribute Limitations**

<b>Attribute</b>	<b>Limitation</b>	<b>Requirement</b>	<b>Default</b>
Data Type	Supported Values = {INT; REAL}	Supported Values = {INT; REAL}	INT
Data Units	Supported Values = {Counts, %, Voltage, Current & Units of the Flow Group} (see Appendix D)	Supported Values = {Counts; %}	Counts
Gain Data Type	Supported Values = {INT; REAL}	Supported Values = {INT; REAL}	INT

#### 6-31.7.4 S-Single Stage Controller Object Instance

##### 6-31.7.4.1 Limitations

**Table 6-31.14 Attribute Limitations**

Attribute	Limitation	Requirement	Default
Data Type	Supported Values = {INT; REAL}	Supported Values = {INT; REAL}	INT
Data Units	Supported Values = {Counts, %, Voltage, Current & Units of the Flow Group} (see Appendix D)	Supported Values = {Counts; % }	Counts
Process Variable	Not accessible over the network. The <i>Process Variable</i> input to this object instance is the value of the S-Analog Sensor object instance <i>Value</i> attribute.	N.A.	N.A.
CV Data Type	Not supported	N.A.	N.A.
Control Variable	Not accessible over the network, The <i>Control Variable</i> output from this object is the value of the S-Analog Actuator object instance <i>Value</i> attribute.	N.A.	N.A.

#### 6-31.8 Defining Device Configuration

Public access to the S-Device Supervisor, S-Analog Sensor, S-Analog Actuator, S-Single Stage Controller, and S-Gas Calibration Objects by the Message Router must be supported for configuration of this device type.

This page is intentionally left blank

## 6-32 Vacuum/Pressure Gauge Device

### Device Type: 1C<sub>Hex</sub>

The objective of this profile is to provide a vacuum or pressure measurement profile, which is inclusive of all technologies used to provide the pressure reading. By use of "gauge" subclasses of the S-Analog Sensor, this profile can apply to Heat Transfer Gauges (Convection, Pirani, Thermocouple), Hot Cathode Ion Gauge, Cold Cathode Ion Gauge or a Diaphragm Gauge. The gauge subclasses provide calibration, control and status attributes unique to each gauge type. The S-Analog Sensor Object provides a pressure value to the Assembly instances delineated in this profile.

#### Combination and Multiple Gauges

This profile has been structured to facilitate use of "Combination" gauges - multiple gauges each covering separate, contiguous ranges of pressure, only one gauge active and providing one Pressure Value at any particular time; and "Multiple" gauges – in which all gauge readings are simultaneously available (but not necessarily valid). In both cases, multiple instances of the S-Analog Sensor Object are used.

### 6-32.1 Object Model

The Object Model in Figure 6-32.4 represents a Vacuum/Pressure Gauge Device.

The table below indicates:

- the object classes present in this device
- whether or not the class is required
- the number of instances present in each class

**Table 6-32.1 Objects Present in a Vacuum/Pressure Gauge Device**

Object Class	Optional/Required	# of Instances
CIP Common Required Objects	Required	See Section 6-2.1
Assembly	Required	At least 1 I/O and 1 Explicit
S-Device Supervisor	Required	1
S-Analog Sensor	1 Subclass required as a minimum See S-Analog Sensor Subclasses in object model above.	1 or more
S-Gas Calibration	Optional	1 or more
Trip Point	Optional	1 ... 8
Discrete Output Point	Optional	1 or more
Analog Output Point	Optional	1 or more

## 6-32.2 Class Subclasses

Each class level subclass defines a unique meaning for an overlapping range of class attribute IDs and/or class service IDs. The range for subclass definitions begins at ID 96 and numbers downward for attributes, and ID 63<sub>hex</sub> and numbers downward for services. The subclass for a given class is identified by the value of its Subclass class attribute.

**Table 6-32.2 S-Analog Sensor Class Level Object Subclasses**

Instance ID	Subclass Name	Subclass ID (Attribute 99 Value)	Required	Function	Restrictions
Class	Instance Selector	01	Conditional*	Data Flow from the Active Instance	None

\* Required for combination gauges only

## 6-32.3 Instance Subclasses

Each instance level subclass defines a unique meaning for an overlapping range of instance attribute IDs and/or instance service IDs. The range for subclass definitions begins at ID 96 and numbers downward for attributes, and ID 63<sub>hex</sub> and numbers downward for services. The subclass for a given instance is identified by the value of its Subclass instance attribute. The following tables identify which object instance IDs are assigned subclasses for this device.

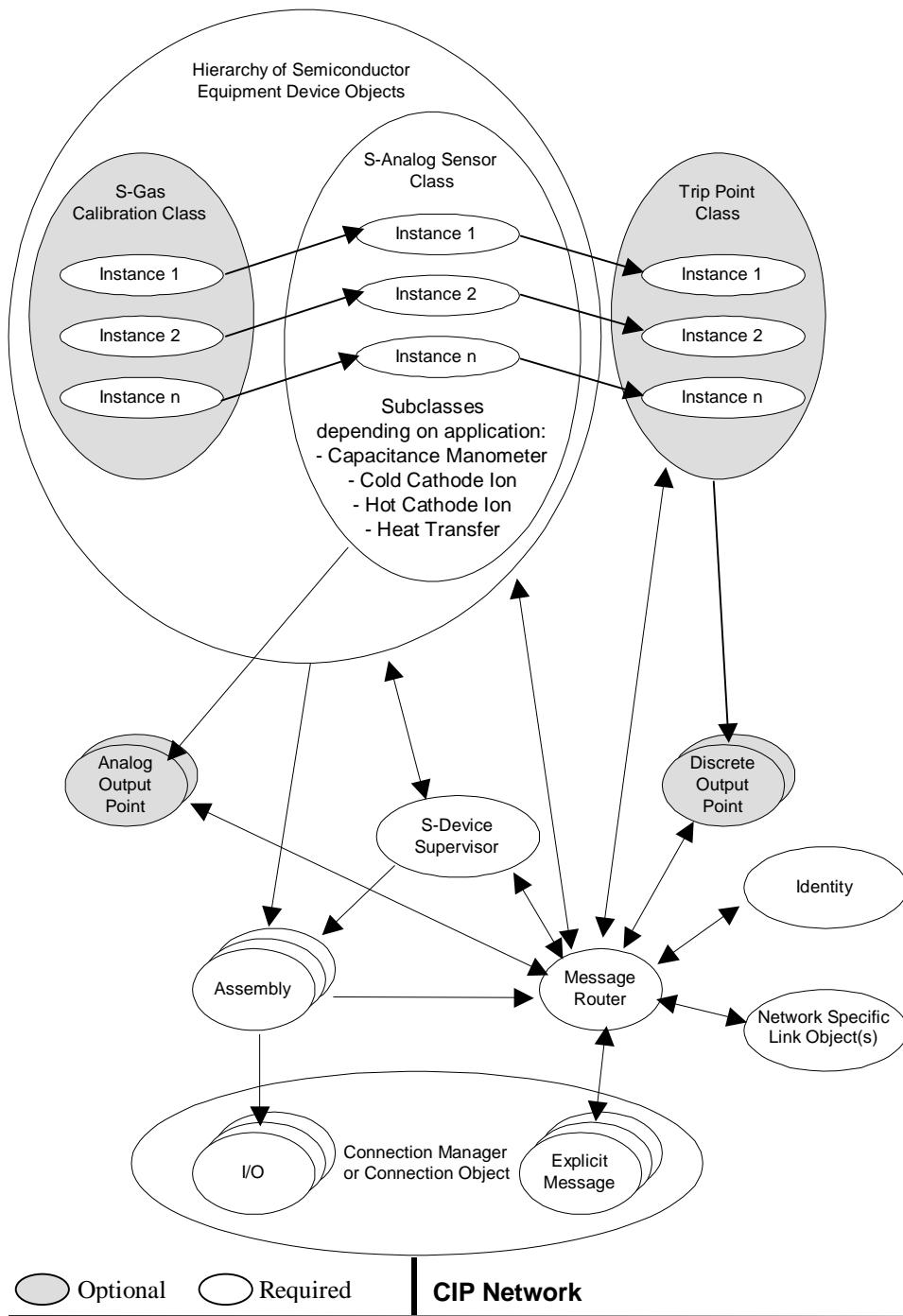
**Table 6-32.3 S-Analog Sensor Object Instance Level Subclasses**

Instance ID	Subclass Name	Subclass ID (Attribute 99 Value)	Required	Function	Restrictions
*	Heat Transfer Vacuum Gauge	02	Conditional **	Gauge Application	None
*	Diaphragm Gauge Gauge	03	Conditional **	Gauge Application	None
*	Cold Cathode Ion Gauge	04	Conditional **	Gauge Application	None
*	Hot Cathode Ion Gauge	05	Conditional **	Gauge Application	None

\* Instance IDs are vendor specific

\*\* The Gauge type Subclass is required if the referenced gauge type is implemented.

Figure 6-32.4 Object Model



## 6-32.4 How Objects Affect Behavior

**Table 6-32.5 Object Effect on Behavior**

Object	Effect on behavior
Identity	Supports the Reset service. Upon receipt of a <i>Reset Service Request</i> of any <i>Type</i> , the Identity Object sends a <i>Reset Service Request</i> to the S-Device Supervisor.
CIP Common Required	See Section 6-2.2 for details.
Assembly	Defines input/output and configuration data format
S-Device Supervisor	Supports the Stop, Start, Reset, Abort, Recover and Perform_Diagnostic services for ALL Application Objects in the device and consolidates the Exception Conditions and Application Objects' Status.  This object behaves differently from the Identity Object in that the S-Device Supervisor object provides a single point of access to the Application Objects only; it does not effect the DeviceNet objects (i.e., Identity, DeviceNet, Connection, etc.).
S-Analog Sensor	Each instance of this object provides a calibrated pressure value from a pressure transducer. This object can also be used to supply an internal potentiometer position used for calibration purposes. Each instance will most likely use a gauge subclass; which subclass is used is determined by the gauge technology used.
S-Gas Calibration	Modifies the correction algorithm of the S-Analog Sensor object. The Gas Calibration Instance attribute of that object specifies which instance is active.
Trip Point	Provides a process trip point comparator for the S-Analog Sensor value. Each instance is linked to an S-Analog Sensor instance. The output of this object may be used to drive the Discrete Output Point object.
Discrete Output Point	Reflects status of Trip Point object instances.
Analog Output Point	Provides an Analog Output from the device which is fed from the S-Analog Sensor value. This analog value may be of a data type and data units different from those of the S-Analog Sensor. This object is required only if the output value is supported as visible to the network.

## 6-32.5 Defining Object Interfaces

**Table 6-32.6 Object Interfaces**

Object	Interface
CIP Common Required	See Section 6-2.3 for details.
Assembly	I/O Connection or Message Router
S-Device Supervisor	Assembly or Message Router
S-Analog Sensor	Assembly or Message Router
S-Gas Calibration	Message Router
Trip Point	Assembly or Message Router
Discrete Output Point	Message Router
Analog Output Point	Message Router

## 6-32.6 I/O Assembly Instances

The manufacturer of a Gauge Device must specify which Assembly instances are supported by the device.

The *S-Analog Sensor* object definition specifies a behavior that modifies the *Data Type* of certain attributes based upon the first valid I/O connection established to an Assembly Object instance. In order to maintain consistency, this device type will only allow connections to either INT or REAL based Assembly instances. Once a valid connection is established, attempts to configure connections, or otherwise access data, to a different type of Assembly instance will return a RESOURCE UNAVAILABLE error.

### Combination Gauges

Though the device supports multiple instances of the S-Analog Sensor class, only a single *Pressure Value* is produced in the Assemblies. The S-Analog Sensor class-level attribute *Active Instance Number* identifies the object instance that is currently active and providing its *Pressure Value* to the *Active Pressure Value* which is, in turn, produced by the input assemblies. Note that this behavior does not apply to the Trip Point and the Analog Output Point objects, which are directly linked to S-Analog Sensor instances.

### Multiple Gauges

Multiple pressure values are available from multiple S-Analog Sensor instances and are provided in each assembly instance that contains multiple *Pressure Value* members indicated by *Pressure Value n*, where *n* corresponds to the S-Analog Sensor instance ID. The maximum number for *n* is specified by the value of the S-Analog Sensor object class attribute *Number of Gauges*. For devices with fewer gauges than the number specified in a given input assembly, the missing Data Components of that assembly will be set to zero (0). Note that for the purpose of bandwidth optimization, I/O connections to such assemblies where less than the included number of instances are valid, the I/O connection produce length can be set accordingly.

The following table identifies the I/O assembly instances. The manufacturer must specify which Assembly instances are supported by the device.

Vacuum/Pressure Gauge Device, Type: 1C<sub>Hex</sub>

Table 6-32.7 I/O Assembly Instances

Instance	Required	Type	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	N	Input	0 - 1	INT Pressure Value							
2	Y (default)	Input	0	Exception Status							
			1 - 2	INT Pressure Value							
3	N	Input	0	Exception Status							
			1	Trip Status Inst. 8	Trip Status Inst. 7	Trip Status Inst. 6	Trip Status Inst. 5	Trip Status Inst. 4	Trip Status Inst. 3	Trip Status Inst. 2	Trip Status Inst. 1
			2 - 3	INT Pressure Value							
4	N	Input	0 - 3	REAL Pressure Value							
5	N	Input	0	Exception Status							
			1 - 4	REAL Pressure Value							
6	N	Input	0	Exception Status							
			1	Trip Status Inst. 8	Trip Status Inst. 7	Trip Status Inst. 6	Trip Status Inst. 5	Trip Status Inst. 4	Trip Status Inst. 3	Trip Status Inst. 2	Trip Status Inst. 1
			2 - 5	REAL Pressure Value							
7	N	Input	0	Exception Status							
			1	Trip Status Inst. 8	Trip Status Inst. 7	Trip Status Inst. 6	Trip Status Inst. 5	Trip Status Inst. 4	Trip Status Inst. 3	Trip Status Inst. 2	Trip Status Inst. 1
8	N	Input	0	Exception Status							
9	N	Input	0 - 1	Active Instance							
			2 - 3	INT Active Pressure Value							
10	N	Input	0	Exception Status							
			1 - 2	Active Instance							
			3 - 4	INT Active Pressure Value							
11	N	Input	0	Exception Status							
			1	Trip Status Inst. 8	Trip Status Inst. 7	Trip Status Inst. 6	Trip Status Inst. 5	Trip Status Inst. 4	Trip Status Inst. 3	Trip Status Inst. 2	Trip Status Inst. 1
			2 - 3	Active Instance							
			4 - 5	INT Active Pressure Value							
12	N	Input	0 - 1	Active Instance							
			2 - 5	REAL Active Pressure Value							
13	N	Input	0	Exception Status							
			1 - 2	Active Instance							
			3 - 6	REAL Active Pressure Value							
14	N	Input	0	Exception Status							
			1	Trip Status Inst. 8	Trip Status Inst. 7	Trip Status Inst. 6	Trip Status Inst. 5	Trip Status Inst. 4	Trip Status Inst. 3	Trip Status Inst. 2	Trip Status Inst. 1
			2 - 3	Active Instance							
			4 - 7	REAL Active Pressure Value							

Vacuum/Pressure Gauge Device, Type: 1C<sub>Hex</sub>

Instance	Required	Type	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
15	N	Input	0 - 1	INT Pressure Value 1							
			2 - 3	INT Pressure Value 2							
			4 - 5	INT Pressure Value 3							
			6 - 7	INT Pressure Value 4							
16	N	Input	0	Exception Status							
			1 - 2	INT Pressure Value 1							
			3 - 4	INT Pressure Value 2							
			5 - 6	INT Pressure Value 3							
			7 - 8	INT Pressure Value 4							
17	N	Input	0	Exception Status							
			1	Trip Status Inst. 8	Trip Status Inst. 7	Trip Status Inst. 6	Trip Status Inst. 5	Trip Status Inst. 4	Trip Status Inst. 3	Trip Status Inst. 2	Trip Status Inst. 1
			2 - 3	INT Pressure Value 1							
			4 - 5	INT Pressure Value 2							
			6 - 7	INT Pressure Value 3							
			8 - 9	INT Pressure Value 4							
18	N	Input	0 - 3	REAL Pressure Value 1							
			4 - 7	REAL Pressure Value 2							
			8 - 11	REAL Pressure Value 3							
			12 - 15	REAL Pressure Value 4							
19	N	Input	0	Exception Status							
			1 - 4	REAL Pressure Value 1							
			5 - 8	REAL Pressure Value 2							
			9 - 12	REAL Pressure Value 3							
			13 - 16	REAL Pressure Value 4							
20	N	Input	0	Exception Status							
			1	Trip Status Inst. 8	Trip Status Inst. 7	Trip Status Inst. 6	Trip Status Inst. 5	Trip Status Inst. 4	Trip Status Inst. 3	Trip Status Inst. 2	Trip Status Inst. 1
			2 - 5	REAL Pressure Value 1							
			6 - 9	REAL Pressure Value 2							
			10 - 13	REAL Pressure Value 3							
			14 - 17	REAL Pressure Value 4							

Vacuum/Pressure Gauge Device, Type: 1C<sub>Hex</sub>

Instance	Required	Type	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
21	N	Input	0 - 1	INT Pressure Value 1							
			2 - 3	INT Pressure Value 2							
			4 - 5	INT Pressure Value 3							
			6 - 7	INT Pressure Value 4							
			8 - 9	INT Pressure Value 5							
			10 - 11	INT Pressure Value 6							
			12 - 13	INT Pressure Value 7							
			14 - 15	INT Pressure Value 8							
22	N	Input	0	Exception Status							
			1 - 2	INT Pressure Value 1							
			3 - 4	INT Pressure Value 2							
			5 - 6	INT Pressure Value 3							
			7 - 8	INT Pressure Value 4							
			9 - 10	INT Pressure Value 5							
			11 - 12	INT Pressure Value 6							
			13 - 14	INT Pressure Value 7							
			15 - 16	INT Pressure Value 8							
23	N	Input	0	Exception Status							
			1	Trip Status Inst. 8	Trip Status Inst. 7	Trip Status Inst. 6	Trip Status Inst. 5	Trip Status Inst. 4	Trip Status Inst. 3	Trip Status Inst. 2	Trip Status Inst. 1
			2 - 3	INT Pressure Value 1							
			4 - 5	INT Pressure Value 2							
			6 - 7	INT Pressure Value 3							
			8 - 9	INT Pressure Value 4							
			10 - 11	INT Pressure Value 5							
			12 - 13	INT Pressure Value 6							
			14 - 15	INT Pressure Value 7							
			16 - 17	INT Pressure Value 8							

Vacuum/Pressure Gauge Device, Type: 1C<sub>Hex</sub>

Instance	Required	Type	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
24	N	Input	0 - 3	REAL Pressure Value 1							
			4 - 7	REAL Pressure Value 2							
			8 - 11	REAL Pressure Value 3							
			12 - 15	REAL Pressure Value 4							
			16 - 19	REAL Pressure Value 5							
			20 - 23	REAL Pressure Value 6							
			24 - 27	REAL Pressure Value 7							
			28 - 31	REAL Pressure Value 8							
25	N	Input	0	Exception Status							
			1 - 4	REAL Pressure Value 1							
			5 - 8	REAL Pressure Value 2							
			9 - 12	REAL Pressure Value 3							
			13 - 16	REAL Pressure Value 4							
			17 - 20	REAL Pressure Value 5							
			21 - 24	REAL Pressure Value 6							
			25 - 28	REAL Pressure Value 7							
			29 - 32	REAL Pressure Value 8							
26	N	Input	0	Exception Status							
			1	Trip Status Inst. 8	Trip Status Inst. 7	Trip Status Inst. 6	Trip Status Inst. 5	Trip Status Inst. 4	Trip Status Inst. 3	Trip Status Inst. 2	Trip Status Inst. 1
			2 - 5	REAL Pressure Value 1							
			6 - 9	REAL Pressure Value 2							
			10 - 13	REAL Pressure Value 3							
			14 - 17	REAL Pressure Value 4							
			18 - 21	REAL Pressure Value 5							
			22 - 25	REAL Pressure Value 6							
			26 - 29	REAL Pressure Value 7							
			30 - 33	REAL Pressure Value 8							

Vacuum/Pressure Gauge Device, Type: 1C<sub>Hex</sub>

Table 6-32.8 Output Assemblies for Hot Cathode Ion Gauges

Inst ID	Req'd	Type	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
200	N	Output	0	Active Degas Filament 8	Active Degas Filament 7	Active Degas Filament 6	Active Degas Filament 5	Active Degas Filament 4	Active Degas Filament 3	Active Degas Filament 2	Active Degas Filament 1
			1	Active Filament 8	Active Filament 7	Active Filament 6	Active Filament 5	Active Filament 4	Active Filament 3	Active Filament 2	Active Filament 1
			2	reserved	reserved	reserved	reserved	reserved	Clear Emission Off Alarm	Set Degas ON/OFF	Set Emission ON/OFF
201	N	Output	0	Active Filament 8	Active Filament 7	Active Filament 6	Active Filament 5	Active Filament 4	Active Filament 3	Active Filament 2	Active Filament 1
			1	reserved	reserved	reserved	reserved	reserved	Clear Emission Off Alarm	Set Degas State ON/OFF	Set Emission State ON/OFF
202	N	Output	0	Active Degas Filament 8	Active Degas Filament 7	Active Degas Filament 6	Active Degas Filament 5	Active Degas Filament 4	Active Degas Filament 3	Active Degas Filament 2	Active Degas Filament 1
			1	Active Filament 8	Active Filament 7	Active Filament 6	Active Filament 5	Active Filament 4	Active Filament 3	Active Filament 2	Active Filament 1
			2-5	Emission Current							
			6	reserved	reserved	reserved	reserved	reserved	Clear Emission Off Alarm	Set Degas State ON/OFF	Set Emission State ON/OFF
203	N	Output	0	Active Filament 8	Active Filament 7	Active Filament 6	Active Filament 5	Active Filament 4	Active Filament 3	Active Filament 2	Active Filament 1
			1-4	Emission Current							
			5	reserved	reserved	reserved	reserved	reserved	Clear Emission Off Alarm	Set Degas State ON/OFF	Set Emission State ON/OFF

If a fault condition occurs (for example overpressure emission off), the state of the ion gauge will be off and the fault condition must be cleared (Clear Emission Off Alarm) before pulling the "Set Emission State" and/or "Set Degas State" bits to zero then one again to light the ion gauge.

Vacuum/Pressure Gauge Device, Type: 1C<sub>Hex</sub>

Table 6-32.9 Output assemblies for Cold Cathode Ion Gauges

Inst ID	Req'd	Type	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
200	N	Output	0	reserved	reserved	reserved	reserved	reserved	Clear Overpressure High Voltage Off Alarm	reserved	Set High Voltage State ON/OFF
201	N	Output	0-3						High Voltage		
			4	reserved	reserved	reserved	reserved	reserved	Clear Overpressure High Voltage Off Alarm	reserved	Set High Voltage State ON/OFF
202	N	Output	0-3						High Voltage		
			4-7						Sensitivity		
			8	reserved	reserved	reserved	reserved	reserved	Clear Overpressure High Voltage Off Alarm	reserved	Set High Voltage State ON/OFF

If a fault condition occurs (for example overpressure emission off), the state of the ion gauge will be off and the fault condition must be cleared (Clear Overpressure Emission Off Alarm) before pulling the "Set High Voltage State" bit to "zero" then "one" again to light the ion gauge.

All of the elemental components listed above follow standard CIP Data Encoding as specified in Appendix C.

### 6-32.6.1 Mapping I/O Assembly Data Attribute Components

The *Data Type* of the Pressure Value attribute below (and the *Data Type* of other objects used in this profile) is based upon the first valid I/O connection established to an Assembly Object instance.

The following table indicates the I/O assembly Data attribute mapping for this Gauge device.

Table 6-32.10 I/O Assembly Data Mapping

Data Component Name	Class	Class Number		Instance Number	Attribute		
			Sub-class		Name	Number	Type
Pressure Value	S-Analog Sensor	31 hex	-	1 – N	Value	6	INT or REAL
Trip Status	Trip Point	35 hex	-	1 - N	Status	7	BOOL
Exception Status	S-Device Supervisor	30 hex	-	1	Exception Status	12	BYTE
Active Instance	S-Analog Sensor	31 hex	-	Class Level	Active Instance Number	95	UINT
Active Pressure Value *	S-Analog Sensor	31 hex	-	Class Level	Active Value	94	INT or REAL

Vacuum/Pressure Gauge Device, Type: 1C<sub>Hex</sub>

Data Component Name	Class	Class Number		Instance Number	Attribute		
			Sub-class		Name	Number	Type
Active Degas Filament	S-Analog Sensor	31 hex	5	1 - N	Active Degas Filament	81	BYTE
Active Filament	S-Analog Sensor	31 hex	5	1 - N	Active Filament	89	BYTE
High Voltage	S-Analog Sensor	31 hex	4	1 - N	High Voltage	91	REAL
Sensitivity	S-Analog Sensor	31 hex	4	1 - N	Sensitivity	92	REAL

Data Component Name	Class	Class Number		Instance Number	Service		
			Sub-class		Service Name	Service Code	Type
Set Degas State On / Off	S-Analog Sensor	31 hex	5	1 - N	Set Degas State	61 hex	BOOL
Set Emission State On / Off	S-Analog Sensor	31 hex	5	1 - N	Set Emission State	62 hex	BOOL
Clear Emission Off Alarm	S-Analog Sensor	31 hex	5	1 - N	Clear Emission Off Alarm	63 hex	BOOL
Set High Voltage State On / Off	S-Analog Sensor	31 hex	4	1 - N	Set High Voltage State	62 hex	BOOL
Clear Overpressure High Voltage Off Alarm	S-Analog Sensor	31 hex	4	1 - N	Clear Overpressure High Voltage Off Alarm	63 hex	BOOL

\* For Combination Gauges only

## 6-32.7 Object Limitations and Specific Definitions

### 6-32.7.1 S-Device Supervisor Object Instance

#### 6-32.7.1.1 Exception Attributes Definition

The following table specifies the data attribute bit mapping for the Device Exception Detail bytes for this device. For more descriptive information, see the definition of the S-Device Supervisor Object Class.

All status and "Sensor" bits originate from the S-Analog Sensor object. The meaning of Sensor Alarm and Sensor Warning is defined by the Gauge Subclass used. See the object specification for the detailed bit mapping. Noted with each entry is the Instance from which the status byte/bit is mapped.

Combination and multiple gauges will have a set of Device Exception details for each gauge. The S-Analog Sensor class attribute *Number of Gauges* is used to determine gauge count.

Any Exception Bit not supported shall be set to zero (0). Note that this profile allows for only one byte of manufacturer exception detail.

Vacuum/Pressure Gauge Device, Type: 1C<sub>Hex</sub>**Table 6-32.11 Exception Detail Warning (Attribute 14) for single gauges**

Data Component	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Device Exception Detail Warning Size	0	0	0	0	0	0	1	1
Device 1 Exception Detail Warning Byte 0	S-Analog Sensor object Instance 1 — Status Extension							
Device 1 Exception Detail Warning Byte 1	S-Analog Sensor object Instance 1 — Sensor Warning — Byte 0							
Device 1 Exception Detail Warning Byte 2	S-Analog Sensor object Instance 1 — Sensor Warning — Byte 1							
Manufacturer Exception Detail Warning Size	0	0	0	0	0	0	0	1
Manufacturer Exception Detail Warning	8 bits defined by Manufacturer							

**Table 6-32.12 Exception Detail Warning (Attribute 14) for combination and multigauges:**

Data Component	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Device Exception Detail Warning Size	0	0	0	X	X	X	X	X
Device 1 Exception Detail Warning Byte 0	S-Analog Sensor object Class Level Status Extension							
Device 1 Exception Detail Warning Byte 1	S-Analog Sensor object Instance 1 — Sensor Warning — Byte 0							
Device 1 Exception Detail Warning Byte 2	S-Analog Sensor object Instance 1 — Sensor Warning — Byte 1							
Device N Exception Detail Warning Byte 0	S-Analog Sensor object Instance N — Sensor Warning — Byte 0							
Device N Exception Detail Warning Byte 1	S-Analog Sensor object Instance N — Sensor Warning — Byte 1							
Manufacturer Exception Detail Warning Size	0	0	0	0	0	0	0	1
Manufacturer Exception Detail Warning	8 bits defined by Manufacturer							

**Table 6-32.13 Exception Detail Alarms (Attribute 13)**

Data Component	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Device Exception Detail (Alarm) Size	0	0	0	0	X	X	X	X
Device 1 Exception Detail (Alarm) Byte 0	S-Analog Sensor object Instance 1 — Sensor Alarm — Byte 0							
Device 1 Exception Detail (Alarm) Byte 1	S-Analog Sensor object Instance 1 — Sensor Alarm — Byte 1							
Combination and Multi-Gauge Devices Only:								
Device N Exception Detail (Alarm) Byte 0	S-Analog Sensor object Instance N — Sensor Alarm — Byte 0							

**Vacuum/Pressure Gauge Device, Type: 1C<sub>Hex</sub>**

Data Component	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Device N Exception Detail (Alarm) Byte 1	S-Analog Sensor object Instance N — Sensor Alarm — Byte 1							
[End Combination and Multi-Gauge Devices Only]								
Manufacturer Exception Detail (Alarm) Size	0	0	0	0	0	0	0	1
Manufacturer Exception Detail (Alarm)	8 bits defined by Manufacturer							

**6-32.7.1.2 Manufacturer's Device Type Attribute Definition**

The following limitations apply:

**Table 6-32.14 Device Type Attribute Limitations**

Attribute ID	Need in implementation	Access Rule	Name	Data Type	Description of Attribute
3	Required	Get	Device Type	SHORT STRING	Supported values: "VG" for single-instance vacuum gauge "CG" for combination gauge "MG" for multiple gauge (ref S-Analog Sensor instance attribute <i>Subclass Number</i> to determine specific gauge type)

**6-32.7.1.3 Behavior**

The behavior of all Pressure/Vacuum Gauge objects are defined by the S-Device Supervisor Object in Chapter 5.

Ion Gauges only – the Emission OFF state will result in the following behavior:

- a) S-Analog Sensor *Value* attribute will revert to the Safe State / Safe Value, if supported. This mimics the non-executing state of the device.
- b) The Trip Point (if supported) *Status* attribute will be set to the unasserted state (*Override*, if supported, = 1).

**6-32.7.2 S-Analog Sensor Object Instance****6-32.7.2.1 Limitations****6-32.7.2.1.1 Object Instance Number Assignment**

The following instance numbering limitations apply:

**Table 6-32.15 Object Instance Number Limitations**

Instance	Description
1 ... 20	Reserved for Pressure Gauges
21 ... 60	Reserved for Interlock-Relay Setting Monitors

### 6-32.7.2.1.2 Data Type, Data Units and Produce Trigger Delta Type

The following attribute limitations apply:

**Table 6-32.16 Data Limitations**

Attribute	Additions / Limitation	Requirements
Data Type	Supported Values = {INT, REAL} Access Rule = Get only, after an assembly instance connection is established	Supported Value = {INT}
Data Units	Supported Values limited to those found in "Group 4 – Pressure" of Data Units Appendix K, and <i>Counts</i> .	Supported Value = {Counts}
Produce Trigger Delta Type	Percent	Percent

All devices are required to support INT Counts. The meaning of Counts is vendor-specific. Which combinations of Data Units and Data Type are supported are vendor-specific. INT Counts and/or REAL Pressure Units are the most likely options. Other combinations (ex. REAL Counts) may not be supported.

Multiple gauges and combination gauges will require multiple instances of the S-Analog Sensor Object, all of which will support the same Data Type and Data Units.

### 6-32.7.2.1.3 Alarm and Warning Hysteresis (attributes 19 and 23)

For gauges with a logarithmic vacuum reading (hot cathode ion gauge, cold cathode ion gauge, and heat transfer gauges) the Alarm and Warning Hysteresis attributes determine the amount (in the unit "percent of the associated alarm/warning value" ) by which the Value must recover to clear an Alarm Condition.

For gauges with a linear vacuum reading (Diaphragm Gauge) the Alarm and Warning Hysteresis attributes determine the amount (absolute) by which the Value must recover to clear an Alarm Condition.

### 6-32.7.2.2 Object Extensions

This profile uses the extensions for the following instance-level subclasses:

**Table 6-32.17 Object Extensions for Instance-level Subclasses**

Gauge Type	Subclass Number
Heat Transfer Vacuum Gauge	02
Diaphragm Gauge	03
Cold Cathode Ion Gauge	04
Hot Cathode Ion Gauge	05

**6-32.7.3 Trip Point Object Instance****6-32.7.3.1 Limitations****6-32.7.3.1.1 Hysteresis (attribute 10)**

For gauges with a logarithmic vacuum reading (hot cathode ion gauge, cold cathode ion gauge, and heat transfer gauges) the Hysteresis attribute determines the amount (in the unit "percent of the associated low/high trip point") by which the Value must recover to clear a trip point condition.

For gauges with a linear vacuum reading (Diaphragm Gauge) the Hysteresis attribute determines the amount (absolute) by which the Value must recover to clear a trip point condition.

**6-32.7.3.1.2 Source (attribute 14)**

Abbreviated EPATH — the *Source* attribute in this device type is abbreviated as follows:

Logical Segment, Instance Only, 8-bit Logical Address (see Appendix C).

The following defaults apply:

Class ID = 31<sub>hex</sub> [**S-Analog Sensor**].  
Attribute ID = 06 [*Value*].

Therefore, the source attribute uses the following encoding:

**Table 6-32.18 Source Attribute Encoding**

24 <sub>hex</sub>	x
-------------------	---

Where x is the Instance ID of the S-Analog Sensor object whose *Value* attribute is the source.

**6-32.7.3.1.3 Destination (attribute 12)**

Abbreviated EPATH — the *Destination* attribute in this device type is abbreviated as follows:

Logical Segment, Instance Only, 8-bit Logical Address (see Appendix C).

The following defaults apply:

Class ID = 09<sub>hex</sub> [**Discrete Output Point**].  
Attribute ID = 03 [*Value*].

Therefore, the Destination attribute uses the following encoding:

**Table 6-32.19 Destination Attribute Encoding**

24 <sub>hex</sub>	x
-------------------	---

Where x is the Instance ID of the Discrete Output Point object whose *Value* attribute is the Destination.

## **6-32.8 Defining Device Configuration**

Public access to the S-Device Supervisor, S-Gas Calibration, and S-Analog Sensor Objects by the Message Router must be supported for configuration of this device type. There is no Parameter Object defined for access to the device type's configuration parameters.

## 6-33 Programmable Logic Controller Device

### Device Type: 0E<sub>hex</sub>

The Programmable Logic Controller Device type defines a device that acts as a **programmable logic controller**. No control functions are included in this profile.

### 6-33.1 Object Model

The table below indicates:

- the object classes present in this device
- whether or not the class is required
- the number of instances present in each class

**Table 6-33.1 Objects Present in a Programmable Logic Controller Device**

Object Class	Optional/Required	# of Instances
CIP Common Required Objects	Required	See Section 6-2.1

The Programmable Logic Controller Device profile cannot specify the definition of the Assembly Object or the type of application objects necessary for device operation. This portion of the device profile must be supplied by the product developer as described earlier in this chapter, Contents of a Device Profile.

### 6-33.2 Defining Object Interfaces

The objects in the Programmable Logic Controller Device have the interfaces listed in the following table:

**Table 6-33.2 Object Interfaces**

Object	Interface
CIP Common Required	See Section 6-2.3 for details.

## **6-34 ControlNet Physical Layer Component Device**

**Device Type: 32 hex**

The ControlNet Physical Layer Component shall not be addressable from the link. This device type shall include repeaters and various media for the ControlNet physical layer.

## 6-35 Process Control Valve Device

### Device Type: 1D<sub>hex</sub>

A Process Control Valve (PCV) is a device that controls a process condition. This can be physically located before or after a process vessel, container or chamber, therefore controlling the entry or exit of fluid / gas to / from a process chamber. It contains a controller that employs closed loop control by receiving a set point and driving the internal actuator such that the process variable is controlled to the set point. The value of the process variable can be obtained either through digital or analog connections. The controller operates in either of two modes. One mode of operation is closed loop process control. The second drives the control valve directly to a position.

### 6-35.1 Object Model

The Object Model in Figure 6-35.2 represents a Process Control Valve Device. The table below indicates:

- the object classes present in this device
- whether or not the class is required
- the number of instances present in each class

**Table 6-35.1 Objects Present in a Process Control Valve Device**

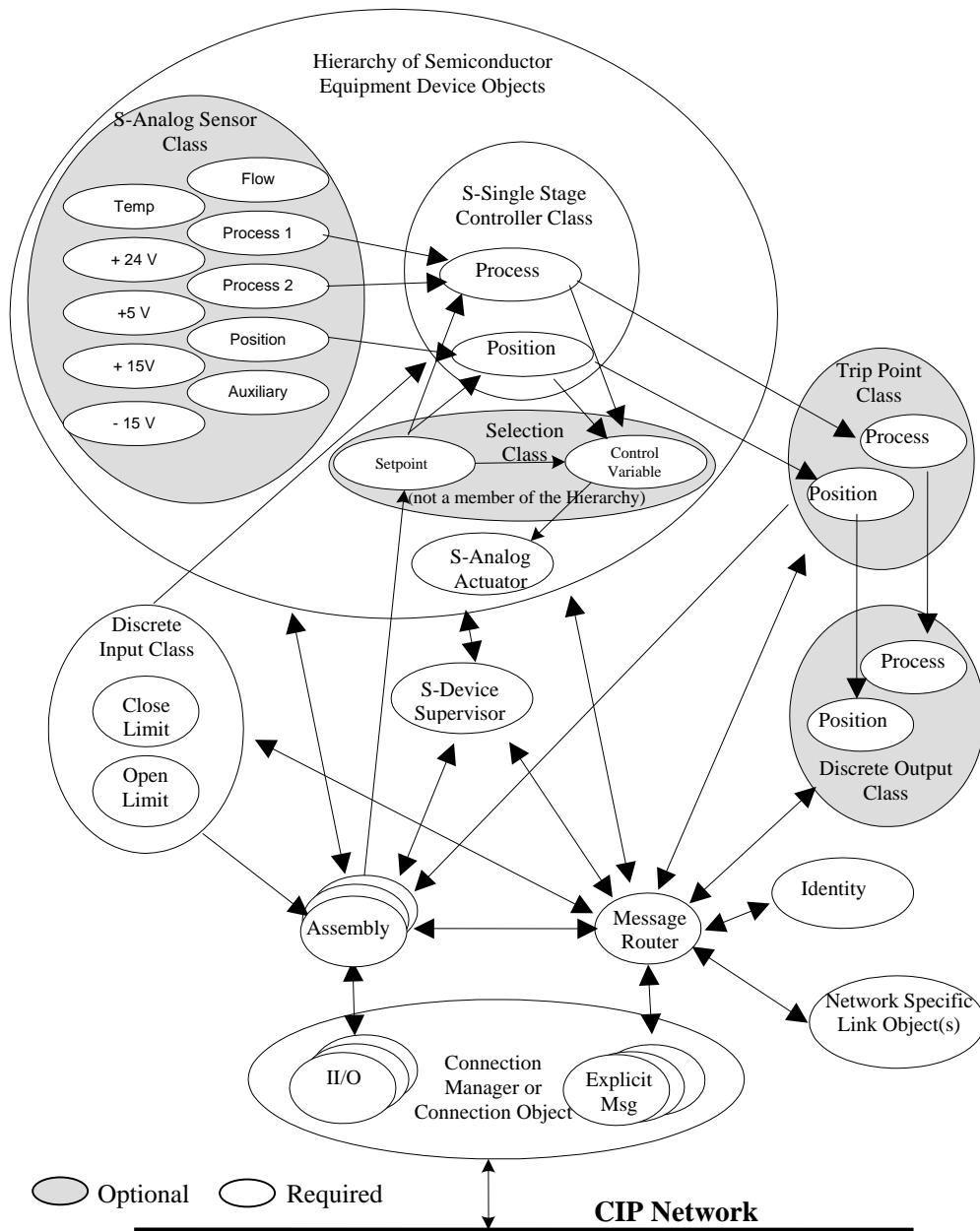
Object Class	Optional/Required	# of Instances
CIP Common Required Objects	Required	See Section 6-2.1
Assembly	Required	at least 2 Input and 2 Output
S-Device Supervisor	Required	1
S-Analog Sensor	Optional	*
S-Analog Actuator	Required	1
S-Single Stage Controller	Required	1 to 2
Selection	Conditional **	2
Trip Point	Optional	1 or more
Discrete Input Point	Optional	2
Discrete Output Point	Optional	1 or more

\* depends on implementation

\*\* Required if two S-Single State Controller Objects are supported.

Process Control Valve Device, Type: 1D<sub>Hex</sub>

Figure 3-35.2 Object Model for the Process Control Valve Device



## 6-35.2 How Objects Affect Behavior

**Table 6-35.3 Object Affect on Behavior**

Object	Effect on behavior
Identity	Supports the Reset service. Upon receipt of a <i>Reset Service Request</i> of any <i>Type</i> , the Identity Object sends a <i>Reset Service Request</i> to the S-Device Supervisor.
CIP Common Required	See Section 6-2.2 for details.
Assembly	Defines input/output and configuration data format
S-Device Supervisor	<p>Supports the Stop, Start, Reset, Abort, Recover and Perform_Diagnostic services for ALL Application Objects in the device and consolidates the Exception Conditions and Application Objects' Status.</p> <p>This object behaves differently from the Identity Object in that the S-Device Supervisor object provides a single point of access to the Application Objects only; it does not affect the DeviceNet specific objects (i.e., Identity, DeviceNet, Connection, etc.).</p>
S-Analog Sensor	Monitors the optional analog process variable input(s), and feeds the process variable value to the Single Stage Controller object. Also, optionally monitors the position of the control valve, as well as other signal levels in the device for diagnostic purposes.
S-Analog Actuator	Drives the physical valve.
S-Single Stage Controller	Calculates the Control Variable required to drive the control valve of the device. Both instances may be present but only one instance is active at any time.
Selection	Provides a means of configuring data flow with the device profile.
Trip Point	Provides a process trip point comparator for the S-Single Stage Controller Process variable value. Each instance is linked to an S-Single Stage Controller instance. The output of this object may be used to drive the Discrete Output Point object.
Discrete Input Point	Monitors the state of the optional limit switches of the device control valve.
Discrete Output Point	Reflect status of Trip point object instances. One Discrete Output Point can be assigned to a Trip Point object.

### 6-35.3 Defining Object Interfaces

**Table 6-35.4 Object Interfaces**

Object	Interface
CIP Common Required	See Section 6-2.3 for details.
Assembly	I/O Connection or Message Router
S-Device Supervisor	Assembly or Message Router
S-Analog Sensor	Assembly or Message Router
S-Analog Actuator	Assembly or Message Router
S-Single Stage Controller	Assembly or Message Router
Selection	Assembly or Message Router
Trip Point	Assembly or Message Router
Discrete Input Point	Assembly or Message Router
Discrete Output Point	Message Router

#### 6-35.3.1 I/O Assembly Instances

The following table identifies the I/O assembly instances supported by the PCV.

The *S-Analog Sensor* and *S-Single Stage Controller* object definition specifies a behavior that modifies the *Data Type* of certain attributes based upon the first valid I/O connection established to an Assembly Object instance. In order to maintain consistency, this device type will only allow connections to either INT or REAL based Assembly instances. Once a valid connection is established, attempts to configure connections, or otherwise access data, to a different type of Assembly instance will return a RESOURCE UNAVAILABLE error.

**Table 6-35.5 I/O Asembly Instances**

Number	Required	Type	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	N	Input	0-1	INT Process Variable							
2	Y (default)	Input	0 1-2	Exception Status Process Variable							
3	N	Input	0 1-2 3-4	Exception Status Process Variable Valve							
4	N	Input	0 1-2 3-4	Exception Status Process Variable Control Setpoint							
5	N	Input	0 1-2 3-4 5-6	Exception Status Process Variable Control Setpoint Valve							
6	N	Input	0 1-2 3-4 5 6-7	Exception Status Process Variable Control Setpoint Control Mode Valve							
7	Y (default)	Output	0-1 2-3	Control Setpoint Control Instance							

**Process Control Valve Device, Type: 1D<sub>Hex</sub>**

Number	Required	Type	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
8	N	Output	0 1-2 3-4	Control Mode Control Setpoint Control Instance							
9	N	Output	0 1-2 3-4 5-6	Control Mode Control Setpoint Control Instance Process Variable							
10	N	Input	0	Exception Status							
11	N	Input	0 1-2 3-4 5	Exception Status Process Variable Valve						Discrete Input 2	Discrete Input 1
12	N	Input	0 1-2 3-4	Exception Status Process Variable Flow							
13	N	Input	0 1-2 3-4 5-6	Exception Status Process Variable Valve Flow							
14	N	Output	1-2	Process Variable							
15	N	Input	0 1-2 3-4	Exception Status Process Variable Temperature							
16	N	Input	0 1-2 3-4 5-6	Exception Status Process Variable Flow Temperature							
17	N	Input	0-3	FP-Process Variable							
18	Y	Input	0 1-4	Exception Status FP-Process Variable							
19	N	Input	0 1-4 5-8	Exception Status FP-Process Variable FP- Valve							
20	N	Input	0 1-4 5-8	Exception Status FP- Process Variable FP- Control Setpoint							
21	N	Input	0 1-4 5-8 9-12	Exception Status FP-Process Variable FP-Control Setpoint FP- Valve							
22	N	Input	0 1-4 5-8 9 10-13	Exception Status FP-Process Variable FP-Control Setpoint Control Mode FP- Valve							
23	Y	Output	0-3 4-5	FP-Control Setpoint Control Instance							
24	N	Output	0 1-4 5-6	Control Mode FP-Control Setpoint Control Instance							
25	N	Output	0 1-4 5-6 7-10	Control Mode FP-Control Setpoint Control Instance FP-Process Variable							

Process Control Valve Device, Type: 1D<sub>Hex</sub>

Number	Required	Type	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
26	N	Input	0 1-4 5-8 9	Exception Status FP-Process Variable FP-Valve						Discrete Input 2	Discrete Input 1
27	N	Input	0 1-4 5-8	Exception Status FP-Process Variable FP-Flow							
28	N	Input	0 1-4 5-8 9-12	Exception Status FP-Process Variable FP-Valve FP-Flow							
29	N	Output	0-3	FP-Process Variable							
30	N	Input	0 1-4 5-9	Exception Status FP-Process Variable FP-Temperature							
31	N	Input	0 1-4 5-8 9-12	Exception Status FP-Process Variable FP-Flow FP-Temperature							
32	N	Output	0-1 2-5 4-9 8-13 12-17	Control Mode FP-Control Setpoint Kp Ki Kd							

The manufacturer of a Process Control Valve Device must specify which optional Assembly instances are supported by the device.

### 6-35.3.2 Mapping I/O Assembly Data Attribute Components

The Data Type of the objects in this profile are based upon the first valid I/O Connection established to an Assembly Object instance. See text of 6-35.4.1.

The following table indicates the I/O assembly Data attribute mapping for this PCV device.

**Table 6-35.6 I/O Assembly Data Mapping**

Data Component Name	Class		Instance Number	Attribute		
	Name	Number		Name	Number	Type
<b>Process Variable</b>	S-Single Stage Controller	33 <sub>hex</sub>	1	Process Variable	7	*
<b>Valve</b>	S-Analog Sensor	31 <sub>hex</sub>	3	Value	6	**
<b>Flow</b>	S-Analog Sensor	31 <sub>hex</sub>	4	Value	6	**
<b>Temperature</b>	S-Analog Sensor	31 <sub>hex</sub>	5	Value	6	**
<b>Control Mode</b>	S-Single Stage Controller	33 <sub>hex</sub>	1	Control Mode	5	USINT
<b>Control Setpoint</b>	Selection	2E <sub>hex</sub>	1	input_data_value	15	*
<b>Control Instance</b>	Selection	2E <sub>hex</sub>	1	destination_used	14	UINT
<b>Status</b>	S-Device Supervisor	30 <sub>hex</sub>	1	Exception Status	12	BYTE
<b>Discrete Input N</b>	Discrete Input Point	08 <sub>hex</sub>	N	Value	3	BOOL
<b>Kd</b>	Derivative Gain	33 <sub>hex</sub>	I	Kd	92	REAL
<b>Ki</b>	Integral Gain	33 <sub>hex</sub>	I	Ki	93	REAL
<b>Kp</b>	Poportional Gain	33 <sub>hex</sub>	I	Kp	94	REAL

\* As specified by S-Single Stage Controller *Data Type* attribute.

\*\* As specified by S-Analog Sensor *Data Type* attribute.

### 6-35.4 Object Limitations and Specific Definitions

#### 6-35.4.1 S-Device Supervisor Object

The following attribute limitations apply:

**Table 6-35.7 Device Type Attribute Limitations**

Attribute	Limitation
Device Type	Supported Values: “PCV” = Process Variable Control Valve device

### 6-35.4.1.1 Exception Attributes Definition

The following table specifies the data attribute bit mapping for the Device Exception Detail bytes for this PCV device. For more descriptive information, refer to the definition of the S-Device Supervisor Object Class (and, in particular, the Semantics of the Exception Detail Alarm and Exception Detail Warning attributes). Noted, for each entry, is the Object and Instance from which the Status byte/bit is mapped. Refer to the object specification for the detailed bit mapping.

Any Exception Bit not supported must default to 0. Note that this profile allows for only one byte of manufacturer specific exception detail.

**Table 6-35.8 Data Attribute Bit Mapping**

Data Component Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
<b>PCV Device Exception Detail Size</b>	0	0	0	0	0	1	0	1
<b>PCV Device Exception Detail Byte #0</b>	Process Variable High S-Analog Sensor (2)	Process Variable Low S-Analog Sensor (2)	Not Reading Valid * S-Analog Sensor (2)	Position Control S-Single Stage Controller (2)	Process Variable Control S-Single Stage Controller (1)	Process Variable High S-Analog Sensor (1)	Process Variable Low S-Analog Sensor (1)	Not Reading Valid * S-Analog Sensor (1)
<b>PCV Device Exception Detail Byte #1</b>	Temp. Low S-Analog Sensor (5)	Not Reading Valid * S-Analog Sensor (5)	Flow High S-Analog Sensor (4)	Flow Low S-Analog Sensor (4)	Not Reading Valid * S-Analog Sensor (4)	Valve High S-Analog Sensor (3)	Valve Low S-Analog Sensor (3)	Not Reading Valid * S-Analog Sensor (3)
<b>PCV Device Exception Detail Byte #2</b>	Not Reading Valid * S-Analog Sensor (8)	Auxiliary Input High S-Analog Sensor (7)	Auxiliary Input Low S-Analog Sensor (7)	Not Reading Valid * S-Analog Sensor (7)	DeviceNet 24V High ** S-Analog Sensor (7)	DeviceNet 24V Low ** S-Analog Sensor (6)	Not Reading Valid * S-Analog Sensor (6)	Temp. High S-Analog Sensor (5)
<b>PCV Device Exception Detail Byte #3</b>	Device -15V High S-Analog Sensor (10)	Device -15V Low S-Analog Sensor (10)	Not Reading Valid * S-Analog Sensor (10)	Device +15V High S-Analog Sensor (9)	Device +15V Low S-Analog Sensor (9)	Not Reading Valid * S-Analog Sensor (9)	Logic 5V High S-Analog Sensor (8)	Logic 5V Low S-Analog Sensor (8)
<b>PCV Device Exception Detail Byte #4</b>	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Status Discrete Input Point (2)	Status Discrete Input Point (1)
<b>Manufacturer Exception Detail Size</b>	0	0	0	0	0	0	0	1

Process Control Valve Device, Type: 1D<sub>Hex</sub>

Data Component Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Manufacturer Exception Detail	8 Bits defined by Manufacturer							

\* Only used in the Warning Exception Detail, this bit is always = 0 in the Alarm Exception Detail. ‘Not’ indicates the use of the negation of ‘Reading Valid’.

\*\* Not supported if CIP network is not DeviceNet. Shall be zero.

#### 6-35.4.2 S-Analog Sensor Object

This section provides the additional components and specifications for this object as it applies to this profile.

The temperature instance is location specific to each manufacturer, if supported. The physical location of the actual temperature sensor the instance reports is determined by the manufacturer.

The following instance limitations apply:

**Table 6-35.9 Instance Limitations**

Instance	Description	Subclass
1	Process Variable Input #1 (Low Range)	06 *
2	Process Variable Input #2 (High Range)	06 *
3	Control Valve Position Feedback	none
4	Flow Sensor	none
5	Temperature Sensor	none
6	DeviceNet 24V Power Input	none
7	Auxiliary Voltage Input	none
8	Logic 5V	none
9	Device +15V	none
10	Device -15V	none

\* S-Analog Sensor Object Instance Subclass 06 - Transfer Function

The following attribute limitations apply:

Process Control Valve Device, Type: 1D<sub>Hex</sub>**Table 6-35.10 Attribute Limitations**

<b>Attribute</b>	<b>Instance</b>	<b>Limitation</b>	<b>Requirements</b>
Data Type	All	Supported Values = {INT; REAL} Access Rule = Get only, after an assembly instance connection is established	INT
Data Units	1,2 3 4 5 6,7,8,9,10	Supported Values = {Counts, Percent, Torr, mTorr, Bar, mBar, Pa, kPa} Supported Values = {Counts, Percent, Degrees} Supported Values = {Counts, Percent, SCCM, SLM} Supported Values = {Counts, Centigrade, Fahrenheit} Supported Values = {Counts, Percent, Volts}	Counts
Offset-A Data Type	All	Supported Values = {INT; REAL}	INT
Gain Data Type	All	Supported Values = {INT; REAL}	INT
Value	All	For: Data Units = Counts Data Type = INT 7FFF = 140% of Full Scale Sensor Reading 0000 = 0 8000 = -140% of Full Scale Sensor Reading	

**6-35.4.3 S-Single Stage Controller Object**

The following instance limitations apply:

**Table 6-35.11 Instance Limitations**

<b>Instance</b>	<b>Description</b>	<b>Subclass</b>
1	Process Variable Control	01 *
2	Control Valve Position	none

\* S-Single Stage Controller Object Instance Subclass 01 - PID and Source Select

The following attribute limitations apply:

**Table 6-35.12 Attribute Limitations**

<b>Attribute</b>	<b>Limitation</b>	<b>Requirements</b>
Data Type	Supported Values = {INT; REAL} Access Rule = Get only, after an assembly instance connection is established	INT
Data Units / Instance 1	Supported Values = {Counts, Percent, Torr, mTorr, Bar, mBar, Pa, kPa}	Counts
Data Units / Instance 2	Supported Values = {Counts, Percent, Degrees} 0 = closed	Counts
Process Variable		Shall be supported

#### 6-35.4.4 Selection Object

The following instance limitations apply:

Where supported, this profile requires the Selection Object Revision 2 or higher.

**Table 6-35.13 Instance Limitations**

Instance	Description
1	Setpoint
2	Control Variable

The following attribute values shall be assigned for Instance 1:

**Table 6-35.14 Attribute Limitations – Instance 1**

Attribute	Limitation (assigned values are shown in hex format)
max_destinations	02
number_of_destinations	02
destination_list	06 / 20 / 33 / 24 / 01 / 30 / 06 — S-Single Stage Controller, Instance 1, setpoint 06 / 20 / 33 / 24 / 02 / 30 / 06 — S-Single Stage Controller, Instance 2, setpoint Access Rule = Get Only
max_sources	00
number_of_sources	00
source_list	Not Supported
source_used	00
algorithm_type	04 — Programmable Data Flow Access Rule = Get Only
object_source_list	00 Access Rule = Get Only
destination_used	Range = 0 - 2

The following attribute values shall be assigned for Instance 2:

This object instance is not able to be configured over the network. Its purpose is to function as an automatic Data Flow selection mechanism.

**Process Control Valve Device, Type: 1D<sub>Hex</sub>****Table 6-35.15 Attribute Limitations – Instance 2**

<b>Attribute</b>	<b>Limitation (assigned values are shown in hex format)</b>
max_destinations	01
number_of_destinations	01
destination_list	06 / 20 / 32 / 24 / 01 / 30 / 06 — S-Analog Actuator, Instance 1, Value Access Rule = Get Only
max_sources	02
number_of_sources	02
source_list	Not Supported
source_used	Range = 1 - 2 Access Rule = Get Only
algorithm_type	04 — Programmable Data Flow Access Rule = Get Only
object_source_list	06 / 20 / 33 / 24 / 01 / 30 / 09 — S-Single Stage Controller, Instance 1, Control Variable 06 / 20 / 33 / 24 / 02 / 30 / 09 — S-Single Stage Controller, Instance 2, Control Variable Access Rule = Get Only
destination_used	01 Access Rule = Get Only

**6-35.4.5 Discrete Input Point Object**

The following instances shall be applied as follows:

**Table 6-35.16 Instances**

<b>Instance</b>	<b>Description</b>
1	Close Limit Switch
2	Open Limit Switch

It is recommended that these object instances support the Off\_On Cycles attribute.

**6-35.5 Defining Device Configuration**

Public access to the S-Device Supervisor, S-Analog Sensor, and S-Single Stage Controller Objects by the Message Router must be supported for configuration of this device type.

For the *Data Units* for the *S-Analog Sensor Object* instances one through four are for process inputs, the *S-Single Stage Controller Object's Setpoint* and the *S-Single Stage Controller Object's Process Variable* will be self-modifying for any change in *Data Units* to any other of these attributes. The self-modified value shall give consistency of *Data Units* to all these instances. This is defined by the class level attributes of Control Insurance and Control Setpoint.

## 6-36 Turbomolecular Vacuum Pump Device

### Device Type: 21<sub>hex</sub>

A Turbomolecular Vacuum Pump (Turbo Pump) is a type of vacuum pump. A vacuum pump is a device that is used for pumping air and other gasses for the purposes of generating negative pressures on its inlet port.

The Turbo Pump device is composed of a motor, a mechanical (turbomolecular) pumping apparatus and various electronic control elements. Typically, thermal management represents another component of the device.

### 6-36.1 Object Model

The Object Model in Figure 6-36.2 represents a Turbo Pump Device. The table below indicates:

- the object classes present in this device
- whether or not the class is required
- the number of instances present in each class

**Table 6-36.1 Objects Present in a Turbo Pump Device**

Object Class	Optional/Required	# of Instances
CIP Common Required Objects	Required	See Section 6-2.1
Assembly	Required	at least 2 Input and 2 Output
S-Device Supervisor	Required	1
Register	Optional	0 or 1
Discrete Input Point	Required	1
Discrete Output Point	Required	1, 2 or 3
AC/DC Drive	Required	1
S-Analog Sensor	Optional	as many as 5
S-Single Stage Controller	Optional	0 or 1

### 6-36.2 Class Subclasses

Each class level subclass defines a unique meaning for an overlapping range of class attribute IDs and/or class service IDs. The range for subclass definitions begins at ID 96 and numbers downward for attributes, and ID 63<sub>hex</sub> and numbers downward for services. The subclass for a given class is identified by the value of its Subclass class attribute.

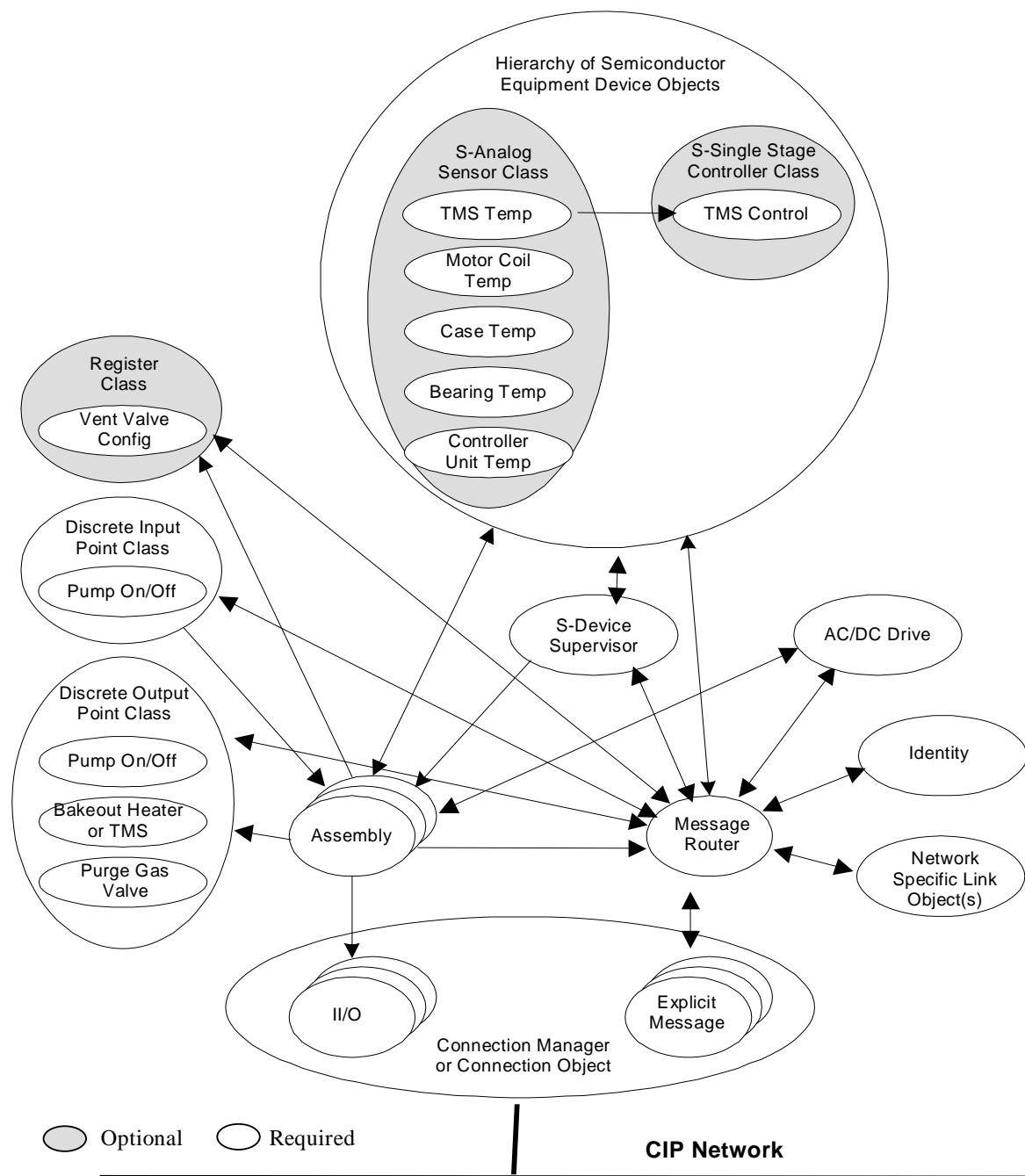
There are no Class Level Subclasses specified for the Turbo Pump device.

### 6-36.3 Instance Subclasses

Each instance level subclass defines a unique meaning for an overlapping range of instance attribute IDs and/or instance service IDs. The range for subclass definitions begins at ID 96 and numbers downward for attributes, and ID 63<sub>hex</sub> and numbers downward for services. The subclass for a given instance is identified by the value of its Subclass instance attribute.

There are no Instance Level Subclasses specified for the Turbo Pump device.

Figure 6-36.2 Object Model for the Turbo Pump Device



## 6-36.4 How Objects Affect Behavior

**Table 6-36.3 Object Affect on Behavior**

Object	Effect on behavior
Identity	Supports the Reset service. Upon receipt of a Reset Service request, a similar request is passed to the S-Device Supervisor.
CIP Common Required	See Section 6-2.2 for details.
Assembly	Defines input/output and configuration data format
S-Device Supervisor	Supports the Stop, Start, Reset, Abort, Recover and Perform_Diagnostic services for ALL Application Objects in the device and consolidates the Exception Conditions and Application Objects' Status
Register	Vent Valve configuration settings
Discrete Input Point	No effect
Discrete Output Point	Turns pump on and off and affects the states of the Bakeout Heater and Purge Valve. The value of Discrete Output Point Object instance 1, Value attribute (ID 3) is sent to AC/DC Drive Object instance 1, Speed Control attribute (ID 38) Bit 0 (Run Request).
AC/DC Drive	Manages the operation of the pump motor. The value of the Speed Status attribute (ID 39) Bit 0 (Running) is sent to Discrete Input Point object instance 1, Value attribute (ID 3).
S-Analog Sensor	Alarm Trip Points may cause the pump to shut down as specified. Feeds values to the Process Variables of S-Single Stage Controllers.
S-Single Stage Controller	Control various temperature zones in the pump assembly

## 6-36.5 Defining Object Interfaces

**Table 6-36.4 Object Interfaces**

Object	Interface
CIP Common Required	See Section 6-2.3 for details.
Assembly	I/O Connection or Message Router
S-Device Supervisor	Assembly or Message Router
Register	Assembly or Message Router
Discrete Input Point	Assembly or Message Router
Discrete Output Point	Assembly or Message Router
AC/DC Drive	Assembly or Message Router
S-Analog Sensor	Assembly or Message Router
S-Single Stage Controller	Assembly or Message Router

## 6-36.6 I/O Assembly Instances

The manufacturer of a Turbo Pump Device must specify which Assembly instances are supported by the device.

The following table identifies the I/O assembly instances.

**Table 6-36.5 I/O Assembly Instances**

Instance	Required	Type	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	Y (default)	Input	0	Exception Status							
			1	Speed Status							
			2	0	0	0	0	0	0	0	Pump On Status
2	Y	Input	0	Exception Status							
			1	Speed Status							
			2	0	0	0	0	0	0	0	Pump On Status
			3 - 4	Pump Speed							
3	N	Input	0	Exception Status							
			1	Speed Status							
			2	0	0	0	0	0	0	0	Pump On Status
			3 - 4	Pump Speed							
			5 - 6	Coil Temp							
			7 - 8	Current							
4	N	Input	0	Exception Status							
			1	Speed Status							
			2	0	0	0	0	0	0	0	Pump On Status
			3 - 4	Pump Speed							
			5 - 6	Inlet Pressure							
5	Y (default)	Output	0	0	0	0	0	0	0	0	Pump On
6	Y	Output	0	0	0	0	0	0	0	0	Pump On
			1	Speed Control							
7	N	Output	0	0	0	0	0	0	0	0	Pump On
			1	Speed Control							
			2 - 3	Speed Target							

**Turbomolecular Vacuum Pump Device, Type: 21<sub>Hex</sub>**

<b>Instance</b>	<b>Required</b>	<b>Type</b>	<b>Byte</b>	<b>Bit 7</b>	<b>Bit 6</b>	<b>Bit 5</b>	<b>Bit 4</b>	<b>Bit 3</b>	<b>Bit 2</b>	<b>Bit 1</b>	<b>Bit 0</b>		
8	N	Output	0	0	0	0	0	0	0	0	Pump On		
			1	Speed Control									
			2 - 3	Pressure Target									
9	N	Output	0	Speed Scale									
			1 - 2	Speed Data Units									
			3 - 4	High Speed Limit									
			5 - 6	Speed Target Data Units									
			7 - 8	Speed Target									
			9	Time Scale									
			10 - 11	Accelerate Time									
			12 - 13	Deceleration Time									
			14 - 15	Speed Trip Time									
			16	Current Scale									
10	N	Output	17 - 18	Current Limit									
			0	Drive Mode									
			1	Process Scale									
			2 - 3	Process Data Units									
11	N	Output	4 - 5	Pressure Target									
			0	Coil Temp Control Mode									
			1 - 2	Coil Temp Setpoint									
			3	Case Temp Control Mode									
			4 - 5	Case Temp Setpoint									
			6	Bearing Temp Control Mode									
12	N	Output	7 - 8	Bearing Temp Setpoint									
			0 - 1	Vent Valve Config									

All of the elemental components listed above follow standard CIP Data Encoding as specified in Appendix C.

Any component that is unsupported shall be set to 0.

## 6-36.7 I/O Assembly Instance Component Mapping

The following table indicates the I/O assembly Data attribute mapping for this device.

**Table 6-36.6 I/O Assembly Data Mapping**

Data Component	Object Name	Class ID	Instance ID	Attribute Name	Attrib	Data Type
<b>Exception Status</b>	S-Device Supervisor	30 <sub>hex</sub>	1	Exception Status	12	BYTE
<b>Pump On Status</b>	Discrete Input Point	08 <sub>hex</sub>	1	Value	3	BOOL
<b>Pump On</b>	Discrete Output Point	09 <sub>hex</sub>	1	Value	3	BOOL
<b>Pump Speed</b>	AC/DC Drive	2A <sub>hex</sub>	1	SpeedActual	7	INT
<b>Speed Status</b>	AC/DC Drive	2A <sub>hex</sub>	1	Speed Status	42	BYTE
<b>Speed Control</b>	AC/DC Drive	2A <sub>hex</sub>	1	Speed Control	41	BYTE
<b>Speed Target</b>	AC/DC Drive	2A <sub>hex</sub>	1	SpeedRef	8	INT
<b>Current</b>	AC/DC Drive	2A <sub>hex</sub>	1	CurrentActual	9	INT
<b>Inlet Pressure</b>	AC/DC Drive	2A <sub>hex</sub>	1	ProcessActual	13	INT
<b>Speed Scale</b>	AC/DC Drive	2A <sub>hex</sub>	1	SpeedScale	22	SINT
<b>Speed Data Units</b>	AC/DC Drive	2A <sub>hex</sub>	1	SpeedActual Data Units	43	ENGUNITS
<b>High Speed Limit</b>	AC/DC Drive	2A <sub>hex</sub>	1	HighSpeed-Limit	21	UINT
<b>Speed Target Data Units</b>	AC/DC Drive	2A <sub>hex</sub>	1	SpeedRef Data Units	44	ENGUNITS
<b>Time Scale</b>	AC/DC Drive	2A <sub>hex</sub>	1	TimeScale	28	SINT
<b>Acceleration Time</b>	AC/DC Drive	2A <sub>hex</sub>	1	AccelTime	18	UINT
<b>Deceleration Time</b>	AC/DC Drive	2A <sub>hex</sub>	1	DecelTime	19	UINT
<b>Overload Time</b>	AC/DC Drive	2A <sub>hex</sub>	1	Speed Trip Time	38	UINT
<b>Current Scale</b>	AC/DC Drive	2A <sub>hex</sub>	1	CurrnetScale	23	SINT
<b>Current Limit</b>	AC/DC Drive	2A <sub>hex</sub>	1	CurrentLimit	10	INT
<b>Drive Mode</b>	AC/DC Drive	2A <sub>hex</sub>	1	DriveMode	6	USINT
<b>Process Scale</b>	AC/DC Drive	2A <sub>hex</sub>	1	ProcessScale	25	SINT
<b>Process Data Units</b>	AC/DC Drive	2A <sub>hex</sub>	1	Process Data Units	45	ENGUNITS
<b>Pressure Target</b>	AC/DC Drive	2A <sub>hex</sub>	1	ProcessRef	14	INT
<b>Coil Temp</b>	S-Analog Sensor	31 <sub>hex</sub>	1	Value	6	INT
<b>Coil Temp Control Mode</b>	S-Single Stage Controller	33 <sub>hex</sub>	1	Control Mode	5	USINT
<b>Coil Temp Setpoint</b>	S-Single Stage Controller	33 <sub>hex</sub>	1	Setpoint	6	INT
<b>Case Temp Control Mode</b>	S-Single Stage Controller	33 <sub>hex</sub>	2	Control Mode	5	USINT
<b>Case Temp Setpoint</b>	S-Single Stage Controller	33 <sub>hex</sub>	2	Setpoint	6	INT
<b>Bearing Temp Control Mode</b>	S-Single Stage Controller	33 <sub>hex</sub>	3	Control Mode	5	USINT
<b>Bearing Temp Setpoint</b>	S-Single Stage Controller	33 <sub>hex</sub>	3	Setpoint	6	INT
<b>Vent Valve Config</b>	Register	07 <sub>hex</sub>	1	Data	4	ARRAY of BOOL

## 6-36.8 Object Limitations and Specific Definitions

This section describes limitations and specific definitions applicable to the listed objects of this device profile when these objects are applied in this type of device.

### 6-36.8.1 S-Device Supervisor

The following attribute limitations apply:

**Table 6-36.7 Device Type Attribute Limitations**

Attribute	Requirements
Device Type	Set to "Turbo Pump" for the Turbomolecular Vacuum Pump Device

The following table specifies the data attribute bit mapping for the **Exception Detail** bytes for this device. For more descriptive information, see the definition of the S-Device Supervisor Object Class.

Any Exception Bit not supported shall default to 0.

**Table 6-36.8 Attribute Bit Mapping**

Data Component	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Turbo Pump Device Exception Detail Size	0	0	0	0	0	0	1	0
Turbo Pump Device Exception Detail Byte 0	Excess Vibration	Startup Timeout *	Speed Trip	Over-current *	Over-speed *	Mains Failure *	TMS Failure	CNT Failure
Turbo Pump Device Exception Detail Byte 1	0	Interlocked	Bearing Fault	Cable Fault *	Controller Overheat	Bearing Overheat	Case Overheat	Motor Coil Overheat
Manufacturer Exception Detail Size **	0	0	0	0	0	0	0	1 **
Manufacturer Exception Detail ***	8 Bits defined by Manufacturer							

\* Exception Detail Alarm only; not supported in Exception Detail Warning

\*\* This example shows a one-byte manufacturer exception detail, but there may be zero or many bytes defined

Turbomolecular Vacuum Pump Device, Type: 21<sub>Hex</sub>**Table 6-36.9 Exception Detail Bit Description**

Exception	Description
CNT Failure	Controller Unit failure
TMS Failure	Temperature Management System failure
Mains Failure	Insufficient power (power failure or incorrect connection)
Over-speed	Pump failure
Over-current	Pump failure
Speed Trip	Pump overload typically sensed by reduced speed exceeding the Speed Trip Time
Startup Timeout	Normal speed from startup not obtained within specified AccelTime
Excess Vibration	Physical fault
Motor Coil Overheat	Thermal fault
Case Overheat	Thermal fault
Bearing Overheat	Thermal fault
Controller Overheat	Thermal fault
Cable Fault	Controller Unit to Pump interconnection cable is misconnected or disconnected
Bearing Fault	Damaged bearing requires service
Interlocked	Normal pump operation is not possible due to an external interlock signal

**6-36.8.2 Register Object****6-36.8.2.1 Object Instance Number Assignment**

The following instance numbering shall apply:

**Table 6-36.10 Instance Numbers**

Instance	Required	Type	Description
1	No	Output	Vent Valve Configuration

**Table 6-36.11 Instance 1—Output—Vent Valve Configuration Data**

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Valve 4		Valve 3		Valve 2		Valve 1	
1	Valve 8		Valve 7		Valve 6		Valve 5	

The following values are defined for each *Valve* listed above.

**Table 6-36.12 Values Defined For Each Valve**

Valve n Config Nibble Value	Description
0	Disabled / No Vent
1	Pump Controlled Vent
2	Vendor Specified Vent
3	Reserved by DeviceNet

### 6-36.8.3 Discrete Input Point Object

#### 6-36.8.3.1 Object Instance Number Assignment

The following instance numbering shall apply:

**Table 6-36.13 Instance Numbering**

Instance	Required	Name	Description
1	Yes	Pump On/Off	0 = Pump Off 1 = Pump On AND Speed > 0

#### 6-36.8.3.2 Attribute Definitions

The following attribute limitations apply:

**Table 6-36.14 Attribute Limitations**

Attribute	Limitation
Off_On Cycles	Supported with Get Only Access Rule

The value of Discrete Input Point Object instance 1, Value attribute (ID 3) is received from AC/DC Drive Object instance ID 1, Speed Status attribute (ID 39) Bit 0 (Running).

### 6-36.8.4 Discrete Output Point Object

#### 6-36.8.4.1 Object Instance Number Assignment

The following instance numbering shall apply:

**Table 6-36.15 Instance Numbering**

Instance	Required	Name	Description of the Value Attribute
1	Yes	Pump On/Off	0 = Pump Off 1 = Pump On
2	No	TMS Function or Bakeout Heater *	0 = TMS or Heater Off 1 = TMS or Heater On
3	No	Purge Gas Valve	0 = Valve Closed 1 = Valve Open

\* TMS or Bakeout Heater is based on Pump Configuration

The value of Discrete Output Point Object instance 1, Value attribute (ID 3) is sent to AC/DC Drive Object instance ID 1, Speed Control attribute (ID 38) Bit 0 (Run Request).

### 6-36.8.5 AC/DC Drive Object

#### 6-36.8.5.1 Attribute Limitations

The following table lists the attributes of the AC/DC Drive Object, Instance 1, and specifies any special definitions for this device profile. Attributes implemented as specified are not shown in this table.

**Table 6-36.16 AC/DC Drive Object, Instance 1 Attributes**

Attribute	Limitation *	Default	Definition
NetRef	Optional Support Access Rule = Get	1	0 = Speed Control NOT by Net 1 = Speed Control by Net
NetProc	Optional Support Access Rule = Get	1	0 = Process Control NOT by Net 1 = Process Control by Net
DriveMode	Supported Values = 0 = Vendor Specified 2 = Closed Loop Speed 4 = Process Control	2	
SpeedActual	Units Specified by Speed Data Units attribute value		
SpeedRef	Units Specified by Speed Data Units attribute value		
ProcessActual	Units Specified by Process Data Units attribute value		Value derived from Gauge at Pump Inlet
ProcessRef	Units Specified by Process Data Units attribute value		
OutputVoltage	Not Supported		
RefFromNet	Optional Support Access Rule = Get	1	0 = Local Speed Reference 1 = Net Speed Reference
ProcFromNet	Optional Support Access Rule = Get	1	0 = Local Process Reference 1 = Net Process Reference
Speed Trip Time	As defined		a Speed Trip Condition results in the setting of the Speed Trip bit of the S-Device Supervisor Exception Detail.
Speed Status	Required		
SpeedActual Data Units	Limited to: Units of Frequency (Volume 1, Appendix D) as well as the following special values: 1007(Hex) = % of Target 0800(Hex) = % of MaxRated	1F0F <sub>hex</sub>	If attribute not supported the value of "RPM" shall be specified.
SpeedRef Data Units	Limited to: Units of Frequency (Volume 1, Appendix D)	1F0F <sub>hex</sub>	If attribute not supported the value of "RPM" shall be specified.
Process Data Units	Limited to Units of Pressure (Volume 1, Appendix D)	1309 <sub>hex</sub>	If attribute not supported the value of "Pa" shall be specified.

\* Vendor may impose further limitations.

### 6-36.8.5.2 Behavior

The value of Discrete Input Point Object instance 1, Value attribute (ID 3) is received from AC/DC Drive Object instance ID 1, Speed Status attribute (ID 39) Bit 0 (Running).

The value of Discrete Output Point Object instance 1, Value attribute (ID 3) is sent to AC/DC Drive Object instance ID 1, Speed Control attribute (ID 38) Bit 0 (Run Request).

The behavior associated with the Speed Trip Condition and the Acceleration Timeout shall be that the respective bit is set in the S-Device Supervisor *exception detail* attribute to indicate the condition. Further, based upon the vendor's declaration of these bits as Critical or Non-Critical, the behavior of the Turbo shall be affected as specified by the vendor's documentation. Typically, either of these conditions will be considered Critical and cause the Turbo to shutdown.

### 6-36.8.6 S-Analog Sensor Object

#### 6-36.8.6.1 Object Instance Number Assignment

The following instance numbering shall apply:

**Table 6-36.17 Instance Number Assignments**

Instance	Description
1	Motor Coil Temperature
2	Case Temperature
3	Bearing Temperature
4	Controller Unit Temperature
5	TMS Temperature

#### 6-36.8.6.2 Attribute Definitions

The following attribute limitations apply to all instances:

**Table 6-36.18 Attribute Limitations**

Attribute	Limitation
Data Type	Supported Value = {INT}
Data Units	Supported Value = {decidegrees Centigrade}

Alarm Trip Points and Warning Trip Points shall be supported and shall be limited to GET only Access if the corresponding components are supported in the *Exception Detail Alarm* and *Exception Detail Warning* attributes defined for the S-Device Supervisor object.

### 6-36.8.7 S-Single Stage Controller Object

#### 6-36.8.7.1 Object Instance Number Assignment

The following instance numbering shall apply:

**Table 6-36.19 Instance Number Assignment**

Instance	Description
1	TMS Control

#### 6-36.8.7.2 Attribute Definitions

The following attribute limitations apply to all instances:

**Table 6-36.20 Attribute Limitations**

Attribute	Limitation
Data Type	Supported Values = {INT}
Data Units	Supported Values = {decidegrees Centigrade}
Control Mode	Supported Values = {Normal, & Off} *

\* The S-Single Stage Controller object instance supports only the Normal and Off control modes as defined by the object specification. The Normal control mode behaves as defined. However, the Off control mode behaves in a manner that is vendor specific. The Vendor shall specify the behavior of the object in the Off control mode. Examples include: maximum cooling, no cooling, or default cooling.

Alarm Trip Points and Warning Trip Points shall be supported and shall be limited to GET only Access if the corresponding components are supported in the *Exception Detail Alarm* and *Exception Detail Warning* attributes defined for the S-Device Supervisor object.

### 6-36.9 Defining Device Configuration

Public access to the S-Device Supervisor by the Message Router must be supported for configuration of this device type.

## 6-37 Residual Gas Analyzer Device

### Device Type: 1E<sub>hex</sub>

A Residual Gas Analyzer (RGA) is a device that measures partial pressures of gasses. The device is composed of a filament, which is controlled through the S-Device Supervisor object, and a partial pressure measurement apparatus, which is controlled by S-Partial Pressure objects.

The S-Partial Pressure objects are instantiated (created), one each, for every Atomic Mass Unit (AMU) to be scanned. Each instance is configured for the measurement of partial pressure or range of partial pressures. Class Level services are supported for requesting scan lists of partial pressures.

### 6-37.1 Object Model

The Object Model in Figure 6-37.2 represents a Residual Gas Analyzer Device. The table below indicates:

- the object classes present in this device
- whether or not the class is required
- the number of instances present in each class

**Table 6-37.1 Objects Present in a Residual Gas Analyzer Device**

Object Class	Optional/Required	# of Instances
CIP Common Required Objects	Required	See Section 6-2.1
Assembly	Required	at least 1 Input and 1 Output
S-Device Supervisor	Required	1
S-Partial Pressure	Required	at least 1 *
S-Analog Sensor	Optional	1
Discrete Input Point	Optional	1
Discrete Output Point	Optional	1 or 2

\* The number of instances is determined by the total number of AMU values or gas species supported per scan. Generally, this number will be 100 - 800.

### 6-37.2 Class Subclasses

Each class level subclass defines a unique meaning for an overlapping range of class attribute IDs and/or class service IDs. The range for subclass definitions begins at ID 96 and numbers downward for attributes, and ID 63<sub>hex</sub> and numbers downward for services. The subclass for a given class is identified by the value of its Subclass class attribute.

There are no Class Level Subclasses specified for the RGA device.

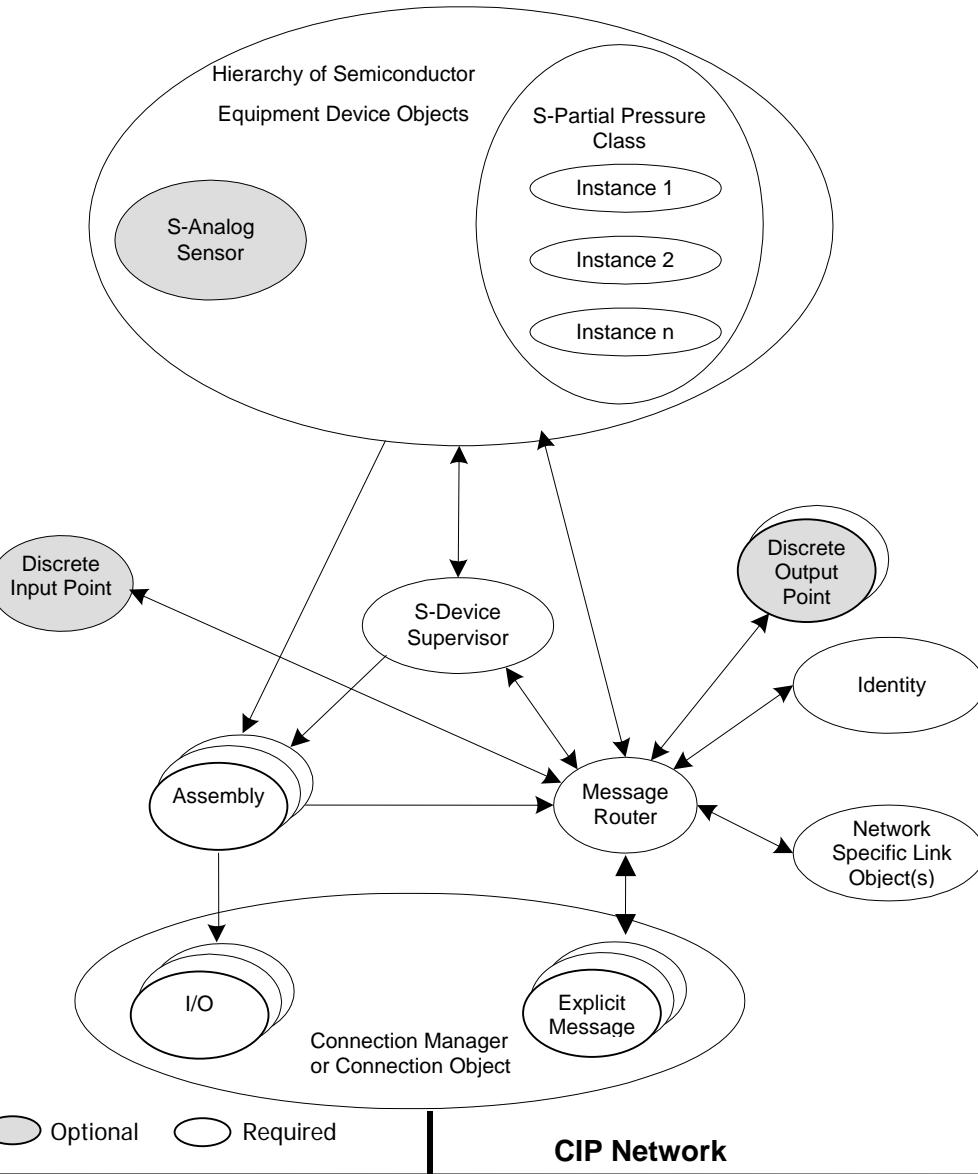
### 6-37.3 Instance Subclasses

Each instance level subclass defines a unique meaning for an overlapping range of instance attribute IDs and/or instance service IDs. The range for subclass definitions begins at ID 96 and numbers downward for attributes, and ID 63<sub>hex</sub> and numbers downward for services. The subclass for a given instance is identified by the value of its Subclass instance attribute.

Residual Gas Analyzer Device, Type: 1E<sub>Hex</sub>

There are no Instance Level Subclasses specified for the RGA device.

**Figure 6-37.2 Object Model for the RGA Device**



## 6-37.4 How Objects Affect Behavior

**Table 6-37.3 Object Affect on Behavior**

Object	Effect on behavior
Identity	Supports the Reset service. Upon receipt of a Reset Service request, a similar request is passed to the S-Device Supervisor.
CIP Common Required	See Section 6-2.2 for details.
Assembly	Defines input/output and configuration data format
S-Device Supervisor	Supports the Stop, Start, Reset, Abort, Recover and Perform_Diagnostic services for ALL Application Objects in the device and consolidates the Exception Conditions and Application Objects' Status
S-Partial Pressure	Provides the means by which Partial Pressures are selected, measured and reported.
S-Analog Sensor	If present, the value from this measurement may interlock the device such that it may not be able to turn on for pressures are that are considered by the manufacturer to be too high
Discrete Input Point	No effect on the device
Discrete Output Point	No effect on the device

## 6-37.5 Defining Object Interfaces

**Table 6-37.4 Object Interfaces**

Object	Interface
CIP Common Required	See Section 6-2.3 for details.
Assembly	I/O Connection or Message Router
S-Device Supervisor	Assembly or Message Router
S-Partial Pressure	Assembly or Message Router
S-Analog Sensor	Assembly or Message Router
Discrete Input Point	Assembly or Message Router
Discrete Output Point	Assembly or Message Router

## 6-37.6 I/O Assembly Instances

The manufacturer of an RGA Device must specify which Assembly instances are supported by the device as well as the maximum number of S-Partial Pressure instances. For devices configured with fewer Partial Pressures than the number specified in a given input assembly, the missing Data Components of that assembly will be set to zero (0). Note that for the purpose of bandwidth optimization, I/O connections to such assemblies, where less than the included number of instances are valid, the I/O connection's *Produced\_Connection\_Size* attribute value can be set accordingly.

Note that the member lists for Partial Pressures and Status attributes are arranged according to Object Instance ID. Even if a given S-Partial Pressure Instance is disabled, it will be included in the I/O Data. Therefore, care must be taken by the user to specify AMU values for each instance in order of desirability, not necessarily AMU value.

The following table identifies the I/O assembly instances.

Residual Gas Analyzer Device, Type: 1E<sub>Hex</sub>

Table 6-37.5 I/O Assembly Instances

Instance	Required	Type	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	Y (default)	Output	0	Filaments On/Off							
2	Y (default)	Input	0	Exception Status							
			1 - 6	Time							
			7 - 10	Partial Pressure 1							
3	Y	Input	0	Exception Status							
			1	Status 1							
4	Y	Input	0	Exception Status							
			1 - 6	Time							
			7 - 10	Partial Pressure 1							
			11 - 14	Partial Pressure 2							
			15 - 18	Partial Pressure 3							
			19 - 22	Partial Pressure 4							
			23 - 26	Partial Pressure 5							
			27 - 30	Partial Pressure 6							
			31 - 34	Partial Pressure 7							
			35 - 38	Partial Pressure 8							
			39 - 42	Partial Pressure 9							
			43 - 46	Partial Pressure 10							
5	Y	Input	0	Exception Status							
			1	Status 1							
			2	Status 2							
			3	Status 3							
			4	Status 4							
			5	Status 5							
			6	Status 6							
			7	Status 7							
			8	Status 8							
			9	Status 9							
			10	Status 10							
6	N	Input	0	Exception Status							
			1 - 6	Time							
			7 - 10	Partial Pressure 1							
			11 - 14	Partial Pressure 2							
			15 - 18	Partial Pressure 3							
			---	---							
			---	---							
			403-406	Partial Pressure 100							

Residual Gas Analyzer Device, Type: 1E<sub>Hex</sub>

Instance	Required	Type	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
7	N	Input	0	Exception Status							
			1	Status 1							
			2	Status 2							
			3	Status 3							
			---	---							
			---	---							
			100	Status 100							
			0	Exception Status							
			1 - 6	Time							
			7 - 10	Partial Pressure 1							
8	N	Input	11 - 14	Partial Pressure 2							
			15 - 18	Partial Pressure 3							
			---	---							
			---	---							
			3203-3206	Partial Pressure 800							
			0	Exception Status							
			1	Status 1							
			2	Status 2							
			3	Status 3							
			---	---							
9	N	Input	---	---							
			---	---							
			800	Status 800							

All of the elemental components listed above follow standard CIP Data Encoding as specified in Appendix C.

### 6-37.6.1 I/O Assembly Instance Component Formats

All of the elemental components listed above follow standard CIP Data Encoding as specified in Appendix C. The encoding order for all data is Low Byte first. All members are LSB aligned and end on Byte boundaries. That is, upper bits are stuffed with 0 as needed to fill to the next byte.

### 6-37.6.2 I/O Assembly Instance Component Mapping

The following table indicates the I/O assembly Data attribute mapping for this device.

**Table 6-37.6 I/O Assembly Data Mapping**

Data Component Name	Object Name	Class ID	Instance ID	Attribute Name	Attribute ID	Member Index	Data Type
<b>Filaments On/Off</b>	S-Partial Pressure	38hex	0 **	Filament Control	32	N.A.	BYTE
<b>Exception Status</b>	S-Device Supervisor	30hex	1	Exception Status	12	N.A.	BYTE
<b>Time</b>	S-Device Supervisor	30hex	1	Time	17	N.A.	DATE_AND_TIME
<b>Partial Pressure i *</b>	S-Partial Pressure	38hex	i *	Partial Pressure	13	N.A.	REAL
<b>Status i *</b>	S-Partial Pressure	38hex	i *	Status	7	N.A.	BYTE

\* The assignment of the Instance ID is a one-for-one mapping to the ordering of the members of the Input Assemblies in the above table. i.e., **Partial Pressure 1 is the Partial Pressure from the S-Partial Pressure Object Instance 1**. Therefore, the user must configure each instance of the S-Partial Pressure object for the desired Partial Pressure in the order they are to appear in the Input Assembly member list.

\*\* Instance 0 corresponds to the Class Level

### 6-37.7 Object Limitations and Specific Definitions

This section describes limitations and specific definitions applicable to the listed objects of this device profile when these objects are applied in this type of device.

**Table 6-37.7 S-Device Supervisor**

Attribute	Requirements
Device Type	Set to "RGA" for Residual Gas Analyzer Device

The following table specifies the data attribute bit mapping for the **Device Exception Detail** bytes for this device. For more descriptive information, see the definition of the S-Device Supervisor Object Class.

Any Exception Bit not supported shall default to 0. Note that this profile allows for only one byte of manufacturer specific exception detail.

Residual Gas Analyzer Device, Type: 1E<sub>Hex</sub>**Table 6-37.8 Attribute Bit Mapping**

Data Component Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
RGA Device Exception Detail Size	0	0	0	0	0	0	1	0
RGA Device Exception Detail Byte 0	0	0	0	0	0	Partial Pressure High	Partial Pressure Low	Reading Invalid
RGA Device Exception Detail Byte 1	0	0	0	0	Filament 4 Fault	Filament 3 Fault	Filament 2 Fault	Filament 1 Fault
Manufacturer Exception Detail Size *	0	0	0	0	0	0	0	1
Manufacturer Exception Detail *	8 Bits defined by Manufacturer							

\* This example shows a one-byte manufacturer exception detail, but there may be zero or many bytes defined

**Table 6-37.9 S-Partial Pressure Object Class**

Attribute	Additions / Limitation	Requirements
Max Instance		Required

**6-37.7.1 S-Analog Sensor Object****6-37.7.1.1 Class**

An RGA device may be able to support an external Capacitance Manometer. However, an implementation may, or may not, include this connection. Therefore, devices that support this object shall support the Class Level services Create and Delete for the S-Analog Sensor object.

**6-37.7.1.2 Instance**

Instance ID 1 (one) is used to interface to an external Capacitance Manometer.

**6-37.7.1.3 Attribute Definitions**

The following attribute limitations apply:

**Table 6-37.10 Attribute Limitations**

Attribute	Additions / Limitation	Requirements
Data Type	Supported Values = {REAL}	
Data Units	Supported Values = {Torr}	
Gain Data Type	Supported Values = {REAL}	
Gain	It is assumed that the external Capacitance Manometer is calibrated to yield a linear output from 0-10 Volts corresponding to Zero-Full Scale reading.	Shall be set to Full Scale (in Torr) of the externally connected Capacitance Manometer divided by 10 (ten).

### 6-37.7.2 Discrete Input Point Object Instance

Instance ID 1 (one) is used to interface to an external Turbomolecular Pump. Specifically, this input is connected to the pump's At Speed indicator. Thus, the *Value* attribute indicates whether the pump is operating at the specified pumping speed.

0 = False / 1 = True.

### 6-37.7.3 Discrete Output Point Object Instance

#### 6-37.7.3.1 Object Instance Number Assignment

The following instance numbering limitations apply:

**Table 6-37.11 Instance Numbering Limitations**

Instance	Description
1	Valve Open - connected externally to a Pump Isolation Valve. Used to command the valve to open or close. 0 = Close / 1 = Open
2	Turbo Pump On - connected externally to a Turbomolecular Pump. Used to command the pump on or off. 0 = Off / 1 = On

## 6-37.8 Defining Device Configuration

Public access to the S-Device Supervisor and S-Partial Pressure Objects by the Message Router must be supported for configuration of this device type. There is no Parameter Object defined for access to the device type's configuration parameters.

## 6-38 DC Power Generator Device

### Device Type: 1F<sub>hex</sub>

A DC Power Generator is a device that converts AC utility power to regulated DC power. It uses sensors to monitor the output, and a controller, which closes the loop with a setpoint such that the output is controlled to the setpoint. The setpoint value can be a voltage, current or power level.

### 6-38.1 Object Model

The Object Model in Figure 6-38.4 represents a DC Power Generator Device. The table below indicates:

- the object classes present in this device
- whether or not the class is required
- the number of instances present in each class

**Table 6-38.1 Objects Present in a DC Power Generator Device**

Object Class	Optional/Required	# of Instances
CIP Common Required Objects	Required	See Section 6-2.1
Assembly	Required	at least 5 Input and 5 Output
S-Device Supervisor	Required	1
S-Analog Sensor	Required	at least 1 as many as 6
S-Analog Actuator	Required	1
S-Single Stage Controller	Required	1 - 3
Selection	Required	1
Register	Required	2

### 6-38.2 Subclasses

Some of the object classes used by this profile define class and/or instance level subclasses. Each subclass defines a unique meaning for an overlapping range of attribute IDs and/or service IDs. The range for subclass definitions begins at ID 96 and numbers downward for attributes, and ID 63<sub>hex</sub> and numbers downward for services. The subclass in use, for a given object class or instance, by the device is identified by the value of its Subclass attribute. The following tables specify the subclass ID and corresponding object class and/or instance ID for this profile.

**DC Power Generator Device, Type: 1F<sub>Hex</sub>****Table 6-38.2 S-Device Supervisor Object Subclass**

<b>Object Instance ID</b>	<b>Subclass Name</b>	<b>Subclass ID (Attribute 99 Value)</b>	<b>Required</b>	<b>Function</b>	<b>Restrictions</b>
1	Power Generator	01	Required	Device Management	None

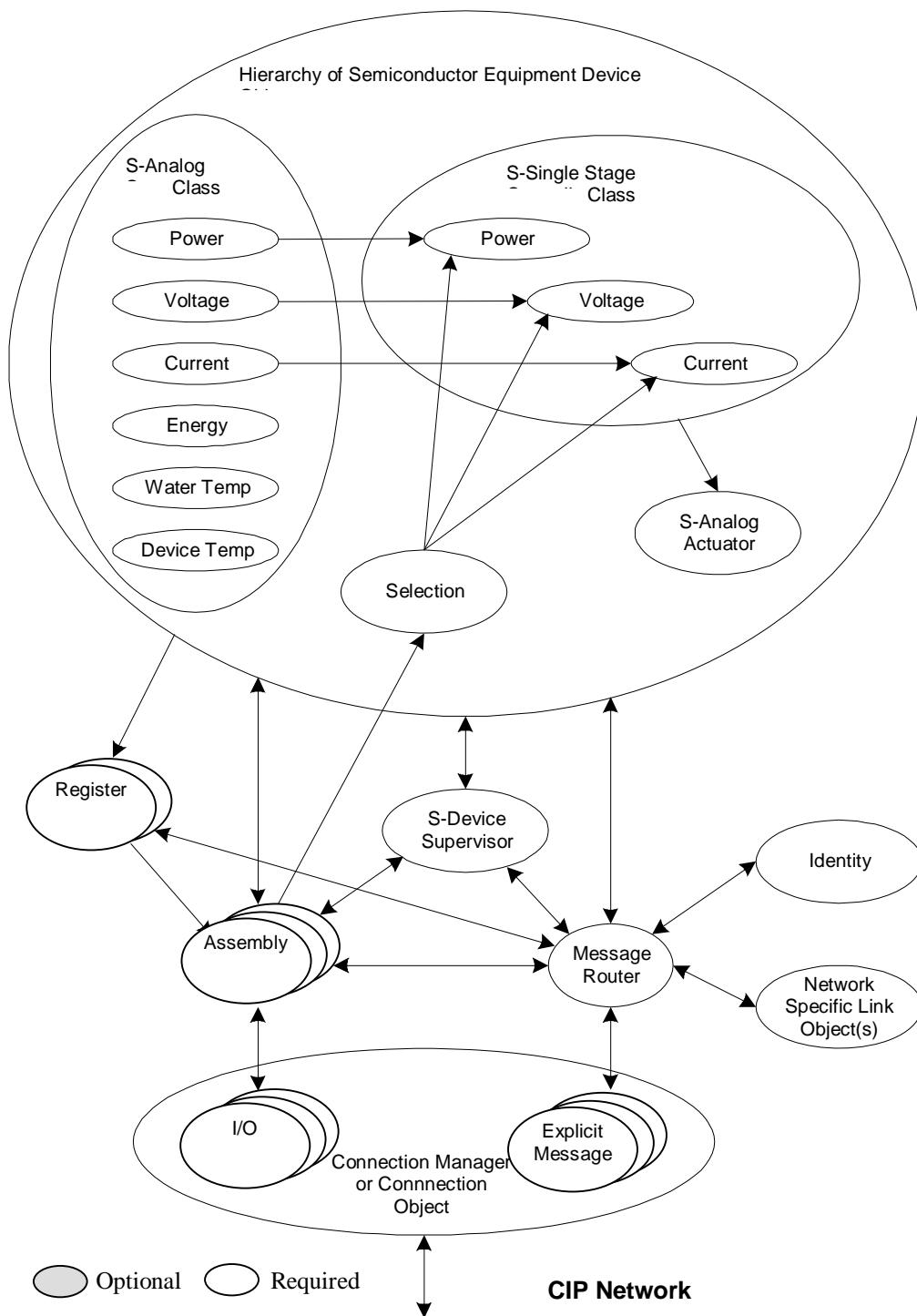
**Table 6-38.3 S-Single Stage Controller Object Subclasses**

<b>Object Instance ID</b>	<b>Subclass Name</b>	<b>Subclass ID (Attribute 99 Value)</b>	<b>Required</b>	<b>Function</b>	<b>Restrictions</b>
1	DC Generator	02	Conditional *	Power Regulation	Data Units = Watts or milliWatts
2	DC Generator	02	Conditional *	Voltage Regulation	Data Units = Volts
3	DC Generator	02	Conditional *	Current Regulation	Data Units = milliAmps

\* At least one of these instances shall be supported

DC Power Generator Device, Type: 1F<sub>Hex</sub>

Figure 6-38.4 Object Model for the DC Power Generator Device



### 6-38.3 How Objects Affect Behavior

**Table 6-38.5 Object Affect on Behavior**

Object	Effect on behavior
Identity	Supports the Reset service. Upon receipt of a <i>Reset Service Request</i> of any <i>Type</i> , the Identity Object sends a <i>Reset Service Request</i> to the S-Device Supervisor.
CIP Common Required	See Section 6-2.2 for details.
Assembly	Defines input/output and configuration data format.
S-Device Supervisor	Supports the Stop, Start, Reset, Abort, Recover and Perform_Diagnostic services for ALL Application Objects in the device and consolidates the Exception Conditions and Application Objects' Status.  This object behaves differently from the Identity Object in that the S-Device Supervisor object provides a single point of access to the Application Objects only; it does not effect the DeviceNet specific objects (i.e., Identity, DeviceNet, Connection, etc.).
S-Analog Sensor	Provides sensed measurements and feeds process variables to the S-Single Stage Controllers.
S-Single Stage Controller	The active instance feeds the control variable to the S-Analog Actuator whereby regulating the output as specified.
S-Analog Actuator	Operates the Power Generator output drive circuitry.
Selection	Activates an instance of the S-Single Stage Controller class and feeds it the Setpoint variable.
Register	Instance 1 - No effect.  Instance 2 - The transition of any interlock bit from 0 to 1 causes an internally generated Stop Service Request to the S-Device Supervisor object instance. Also, any interlock bit in the 1 state shall prevent the S-Device Supervisor object from the EXECUTING state. See Register section below.

### 6-38.4 Defining Object Interfaces

**Table 6-38.6 Object Interfaces**

Object	Interface
CIP Common Required	See Section 6-2.3 for details.
Assembly	I/O Connection or Message Router
S-Device Supervisor	Assembly or Message Router
S-Analog Sensor	Assembly or Message Router
S-Analog Actuator	Assembly or Message Router
S-Single Stage Controller	Assembly or Message Router
Selection	Assembly or Message Router
Register	Assembly or Message Router

## 6-38.5 I/O Assembly Instances

The following table identifies the I/O assembly instances. The manufacturer must specify which Assembly instances are supported by the device.

Any element not supported must be set to zero (0).

**Table 6-38.7 I/O Assembly Instances**

Instance	Required	Type	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0				
1	Y (Default)	Output	0	Power Out 12 (low byte)											
			1	0	0	0	0	Power Out 12 (high nibble)							
			2 - 3	Power Ramp											
			4	0	0	0	0	0	0	0	On				
2	Y (Default)	Input	0	Power In 12 (low byte)											
			1	0	0	0	0	Power In 12 (high nibble)							
			2	Voltage In 12 (low byte)											
			3	0	0	0	0	Voltage In 12 (high nibble)							
			4	Current In 12 (low byte)											
			5	0	0	0	0	Current In 12 (high nibble)							
			6 - 7	Power Ramp											
			8	Op Status											
3	Y	Output	0	0	0	0	0	0	0	0	On				
4	N	Output	0 - 3	Power Limit											
			4 - 7	Voltage Limit											
			8 - 11	Current Limit											
5	Y	Output	0 - 1	Regulation Select											
			2 - 5	Setpoint											
			6 - 7	Power Ramp											
			8	0	0	0	0	0	0	0	On				
6	N	Output	0 - 1	Regulation Select											
			2 - 5	Setpoint											
			6	0	0	0	0	0	0	0	On				
7	Y	Output	0 - 3	Power Out											
			4	0	0	0	0	0	0	0	On				
8	Y	Output	0 - 3	Power Out											
			4 - 5	Power Ramp											
			6	0	0	0	0	0	0	0	On				
9	Y	Input	0	Ex Status											
			1 - 4	Power In											
			5 - 8	Voltage In											
			9 - 12	Current In											
10	N	Input	0	Ex Status											
			1 - 4	Energy In											

DC Power Generator Device, Type: 1F<sub>Hex</sub>

Instance	Required	Type	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
11	N	Input	0	Ex Status							
			1 - 4	Power In							
			5 - 8	Voltage In							
			9 - 12	Current In							
			13 - 16	Energy In							
12	N	Input	0	Ex Status							
			1 - 2	W Temp							
			3 - 4	D Temp							
13	N	Output	0 - 3	Voltage Out							
			4	0	0	0	0	0	0	0	On
14	N	Output	0 - 3	Voltage Out							
			4 - 5	Voltage Ramp							
			6	0	0	0	0	0	0	0	On
15	N	Output	0 - 3	Current Out							
			4	0	0	0	0	0	0	0	On
16	N	Output	0 - 3	Current Out							
			4 - 5	Current Ramp							
			6	0	0	0	0	0	0	0	On
17	N	Input	0	Ex Status							
			1 - 4	Power In							
18	N	Input	0	Ex Status							
			1 - 4	Voltage In							
19	N	Input	0	Ex Status							
			1 - 4	Current In							
20	Y	Input	0	Ex Status							
21	Y	Input	0	Exception Detail Alarm 0 (size, common = 2)							
			1	Exception Detail Alarm 1 (common 0)							
			2	Exception Detail Alarm 2 (common 1)							
			3	Exception Detail Alarm 3 (size, device = 2)							
			4	Exception Detail Alarm 4 (device 0)							
			5	Exception Detail Alarm 5 (device 1)							
			6	Exception Detail Alarm 6 (size, manufacturer = 1)							
			7	Exception Detail Alarm 7 (manufacturer 0)							
22	Y	Input	0	Exception Detail Warning 0 (size, common = 2)							
			1	Exception Detail Warning 1 (common 0)							
			2	Exception Detail Warning 2 (common 1)							
			3	Exception Detail Warning 3 (size, device = 2)							
			4	Exception Detail Warning 4 (device 0)							
			5	Exception Detail Warning 5 (device 1)							
			6	Exception Detail Warning 6 (size, manufacturer = 1)							
			7	Exception Detail Warning 7 (manufacturer 0)							

DC Power Generator Device, Type: 1F<sub>Hex</sub>

Instance	Required	Type	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
23	N	Input	0	Ex Status							
			1	Exception Detail Alarm 0 (size, common = 2)							
			2	Exception Detail Alarm 1 (common 0)							
			3	Exception Detail Alarm 2 (common 1)							
			4	Exception Detail Alarm 3 (size, device = 2)							
			5	Exception Detail Alarm 4 (device 0)							
			6	Exception Detail Alarm 5 (device 1)							
			7	Exception Detail Alarm 6 (size, manufacturer = 1)							
			8	Exception Detail Alarm 7 (manufacturer 0)							
			9	Exception Detail Warning 0 (size, common = 2)							
			10	Exception Detail Warning 1 (common 0)							
			11	Exception Detail Warning 2 (common 1)							
			12	Exception Detail Warning 3 (size, device = 2)							
			13	Exception Detail Warning 4 (device 0)							
			14	Exception Detail Warning 5 (device 1)							
			15	Exception Detail Warning 6 (size, manufacturer = 1)							
			16	Exception Detail Warning 7 (manufacturer 0)							
24	N	Input	0	Op Status							
25	N	Input	0	Interlock Status (low byte)							
			1	Interlock Status (high byte)							
26	N	Input	0	Ex Status							
			1	Op Status							
			2	Interlock Status (low byte)							
			3	Interlock Status (high bytes)							

**6-38.5.1 I/O Assembly Instance Component Formats**

All of the elemental components listed above follow standard CIP Data Encoding as specified in Appendix C. The encoding order for all data is Low Byte first. All members are LSB aligned and end on Byte boundaries. That is, upper bits shall be stuffed with 0 as needed to fill to the next byte.

### 6-38.5.2 I/O Assembly Instance Component Mapping

Each of the *S-Analog Sensor*, *S-Analog Actuator* and *S-Single Stage Controller* object definitions specify a behavior that modifies the *Data Type* of certain attributes based upon the first valid I/O connection established to an Assembly Object instance. See these object definitions for more detail.

The following table indicates the I/O assembly Data attribute mapping for this device.

**Table 6-38.8 I/O Assembly Data Mapping**

Data Component	Object Name	Class ID	Instance ID	Attribute Name	Attribute ID	Data Type
<b>Power Out 12</b>	S-Single Stage Controller	33 <sub>hex</sub>	1	Setpoint	6	UINT
<b>Power In 12</b>	S-Analog Sensor	31 <sub>hex</sub>	1	Value	6	UINT
<b>Voltage In 12</b>	S-Analog Sensor	31 <sub>hex</sub>	2	Value	6	UINT
<b>Current In 12</b>	S-Analog Sensor	31 <sub>hex</sub>	3	Value	6	UINT
<b>Power Ramp</b>	S-Single Stage Controller	33 <sub>hex</sub>	1	Ramp Rate Increment	96	UINT
<b>Voltage Ramp</b>	S-Single Stage Controller	33 <sub>hex</sub>	2	Ramp Rate Increment	96	UINT
<b>Current Ramp</b>	S-Single Stage Controller	33 <sub>hex</sub>	3	Ramp Rate Increment	96	UINT
<b>On</b>	S-Device Supervisor	30 <sub>hex</sub>	1	Output Power Enable	96	BOOL
<b>Op Status</b>	Register	07 <sub>hex</sub>	1	Data	4	ARRAY of 8 BITS
<b>Power Limit</b>	S-Single Stage Controller	33 <sub>hex</sub>	1	Output Limit	86	DINT
<b>Voltage Limit</b>	S-Single Stage Controller	33 <sub>hex</sub>	2	Output Limit	86	DINT
<b>Current Limit</b>	S-Single Stage Controller	33 <sub>hex</sub>	3	Output Limit	86	DINT
<b>Regulation Select</b>	Selection	2E <sub>hex</sub>	1	destination_used	14	UINT
<b>Setpoint</b>	Selection	2E <sub>hex</sub>	1	input_data_value	15	DINT
<b>Power Out</b>	S-Single Stage Controller	33 <sub>hex</sub>	1	Setpoint	6	DINT
<b>Voltage Out</b>	S-Single Stage Controller	33 <sub>hex</sub>	2	Setpoint	6	DINT

**DC Power Generator Device, Type: 1F<sub>Hex</sub>**

Data Component	Object Name	Class ID	Instance ID	Attribute Name	Attribute ID	Data Type
<b>Current Out</b>	S-Single Stage Controller	33 <sub>hex</sub>	3	Setpoint	6	DINT
<b>Power In</b>	S-Analog Sensor	31 <sub>hex</sub>	1	Value	6	DINT
<b>Voltage In</b>	S-Analog Sensor	31 <sub>hex</sub>	2	Value	6	DINT
<b>Current In</b>	S-Analog Sensor	31 <sub>hex</sub>	3	Value	6	DINT
<b>Energy In</b>	S-Analog Sensor	31 <sub>hex</sub>	4	Value	6	DINT
<b>W Temp</b>	S-Analog Sensor	31 <sub>hex</sub>	5	Value	6	INT
<b>D Temp</b>	S-Analog Sensor	31 <sub>hex</sub>	6	Value	6	INT
<b>Interlock Status</b>	Register	07 <sub>hex</sub>	2	Data	4	ARRAY of 16 BITS
<b>Ex Status</b>	S-Device Supervisor	30 <sub>hex</sub>	1	Exception Status	12	BYTE
<b>Exception Alarm</b>	S-Device Supervisor	30hex	1	Exception Detail Alarm	13	STRUCT
<b>Exception Warning</b>	S-Device Supervisor	30hex	1	Exception Detail Warning	14	STRUCT

**6-38.6 Object Limitations and Specific Definitions**

This section describes limitations and specific definitions applicable to the listed objects of this device profile when these objects are applied in this type of device.

**6-38.6.1 S-Device Supervisor**

The following attribute limitations apply:

**Table 6-38.9 Device Type Attribute Limitation**

Attribute	Requirements
Device Type	Set to "DCG" for DC Power Generator Device

The following table specifies the data attribute bit mapping for the **Device Exception Detail** bytes for this device. For more descriptive information, see the definition of the S-Device Supervisor Object Class.

**DC Power Generator Device, Type: 1F<sub>Hex</sub>**

Any Exception Bit not supported shall default to 0. Note that this profile allows for only one byte of manufacturer specific exception detail.

**Table 6-38.10 Data Attribute Bit Mapping**

Data Component	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
<b>Device Exception Detail Size</b>	0	0	0	0	0	0	1	0
<b>Device Exception Detail Byte 0</b>	Reserved 0	Input Power V Low	Input Power V High *	End of Target Life *	Cooling Failure	Device Temp High	Interlock Open	Control Circuit Fault
<b>Byte 1</b>	Reserved 0	Reserved 0	Reserved 0	Reserved 0	Reserved 0	Arc Limit Exceeded *	Process V Low	Regulation
<b>Manufacturer Exception Detail Size</b>	0	0	0	0	0	0	0	1
<b>Manufacturer Exception Detail Byte 0</b>	8 Bits defined by Manufacturer							

\* Only used in the Warning Exception Detail, this bit is always = 0 in the Alarm Exception Detail

**6-38.6.2 S-Analog Sensor Object Instances****Table 6-38.11 Object Instances**

Object Instance ID	Function	Required	Limitations and Specific Definitions
1	Output Power	Required	Supported Data Type and Data Units pairs: DINT / Watts; DINT / milliWatts UINT / % *
2	Output Voltage	Optional	Default Full Scale = 1.0 KV Supported Data Type and Data Units pairs: DINT / Volts (rms); UINT / % *
3	Output Current	Optional	Supported Data Type and Data Units pairs: DINT / milliAmps (rms); UINT / % *
4	Delivered Energy since Output Power was last turned On	Optional	DINT / Joules
5	Cooling Water Temp	Optional	INT / °C Resolution = 1 Degree
6	Device Temp	Optional	INT / °C Resolution = 1 Degree

\* Range = 0 - 4095 corresponding to 0 - 100%

### 6-38.6.3 S-Analog Actuator Object Instances

**Table 6-38.12 Object Instances**

Object Instance ID	Function	Required	Limitations and Specific Definitions
1	Output Power	Required	Supported Data Type and Data Units pairs: DINT / Watts; DINT / milliWatts UINT / % *

\* Range = 0 - 4095 corresponding to 0 - 100%

### 6-38.6.4 S-Single Stage Controller Object Instances

**Table 6-38.13 Object Instances**

Object Instance ID	Function	Required	Limitations and Specific Definitions
1	Output Power Regulation	Optional	Supported Data Type and Data Units pairs: DINT / Watts; DINT / milliWatts; UINT / % *
2	Output Voltage Regulation	Optional	Supported Data Type and Data Units pairs: DINT / Volts (rms); UINT / % *
3	Output Current Regulation	Optional	Supported Data Type and Data Units pairs: DINT / milliAmps (rms); UINT / % *

\* Range = 0 - 4095 corresponding to 0 - 100%

#### 6-38.6.4.1 Behavior

Of instances 1, 2 and 3, only one is active at any given time. Which instance is active is determined by the value of the *destination\_used* attribute of the Selection object instance. In typical operation, the *destination\_used* attribute is mapped to, and hence modified by, the Regulation Select member of an Output Assembly Instance, which in turn, is referenced by the *Consume Path* of the first valid I/O connection established by the device.

### 6-38.6.4.2 Selection Object, Instance 1

Table 6-38.14 Selection Object – Instance 1 Attribute Limitations

Attribute	Limitation (assigned values are shown in hex format)
max_destination	03
number_of_destinations	03
destination_list	06 / 20 / 33 / 24 / 01 / 30 / 06 — S-Single Stage Controller, Instance 1, Setpoint 06 / 20 / 33 / 24 / 02 / 30 / 06 — S-Single Stage Controller, Instance 2, Setpoint 06 / 20 / 33 / 24 / 03 / 30 / 06 — S-Single Stage Controller, Instance 3, Setpoint Access Rule = Get Only
max_sources	00
number_of_sources	00
source_list	Not Supported
source_used	00
algorithm_used	04 - Programmable Data Flow Access Rule = Get Only
object_source_list	00 Access Rule = Get Only
destination_used	Range = 0 - 3

### 6-38.6.5 Register Object

#### 6-38.6.5.1 Register Object, Instance 1 (Input) Operational Status

Table 6-38.15 Data Attribute (8 Bits)

Bit	Description
0	Power On Status (1=on / 0=off)
1	Setpoint Status (1=at setpoint / 0=not)
2	Temp Status (1=OK / 0=overtemp)
3	Arc Detected (1=on / 0=off)
4	Interlock Status Flag (1=OK / 0=open)
5	Reserved for Future Use (0)
6	Reserved for Future Use (0)
7	Reserved for Future Use (0)

### 6-38.6.5.2 Register Object, Instance 2 (Input) Interlock Status

Table 6-38.16 Data Attribute (16 Bits)

Bit	Description
0	Device Not Ready
1	Communications Fault
2	Device State Fault
3	Device Not Configured
4	External
5	Head Cable
6	Head Cover
7	High-Voltage Cable
8	Bias Supply
9	DC Bus Voltage
10	Fan Failure
11	Magnet Temp
12	Power Monitor
13	Cover
14	Reserved for Future Use (0)
15	Reserved for Future Use (0)

1 = Interlocked / 0 = Not Interlocked

### 6-38.6.5.3 Behavior

The transition of any interlock bit from 0 to 1 shall cause an internally generated Stop Service Request to the S-Device Supervisor object instance thereby causing a transition to the IDLE state and disabling the output power of the device. Also, if any interlock bit has a value of 1, the S-Device Supervisor object instance shall not transition to the EXECUTING State: (a) a Start Service Request shall return a Device\_State\_Conflict error, and (b) a Receipt of First Valid I/O Data event shall not cause a state transition.

## 6-38.7 Defining Device Configuration

Public access to the S-Device Supervisor and the application objects by the Message Router must be supported for configuration of this device type.

## 6-39 RF Power Generator Device

### Device Type: 20<sub>hex</sub>

An RF Power Generator is a device that converts AC utility power to regulated RF power. It uses sensors to monitor the output, and a controller, which closes the loop with a setpoint such that the output is controlled to the setpoint. The setpoint value can be a forward or delivered power level. The output frequency can be fixed, set with a user setpoint, or automatically tunable by a controller, which attempts to minimize reflected power.

### 6-39.1 Object Model

The Object Model in Figure 6-39.4 represents an RF Power Generator Device. The table below indicates:

- the object classes present in this device
- whether or not the class is required
- the number of instances present in each class

**Table 6-39.1 Objects Present in a RF Power Generator Device**

Object Class	Optional/Required	# of Instances
CIP Common Required Objects	Required	See Section 6-2.1
Assembly	Required	at least 4 Input and 2 Output
S-Device Supervisor	Required	1
S-Analog Sensor	Required	at least 3 as many as 15
S-Analog Actuator	Required	at least 2 as many as 4
S-Single Stage Controller	Optional	1 - 3
Selection	Required	1
Register	Required	2

### 6-39.2 Subclasses

Some of the object classes used by this profile define class and/or instance level subclasses. Each subclass defines a unique meaning for an overlapping range of attribute IDs and/or service IDs. The range for subclass definitions begins at ID 96 and numbers downward for attributes, and ID 63<sub>hex</sub> and numbers downward for services. The subclass in use, for a given object class or instance, by the device is identified by the value of its Subclass attribute. The following tables specify the subclass ID and corresponding object class and/or instance ID for this profile.

**Table 6-39.2 S-Device Supervisor Object Subclass**

Object Instance ID	Subclass Name	Subclass ID (Attribute 99 Value)	Required	Function	Restrictions
1	Power Generator	01	Required	Device Management	None

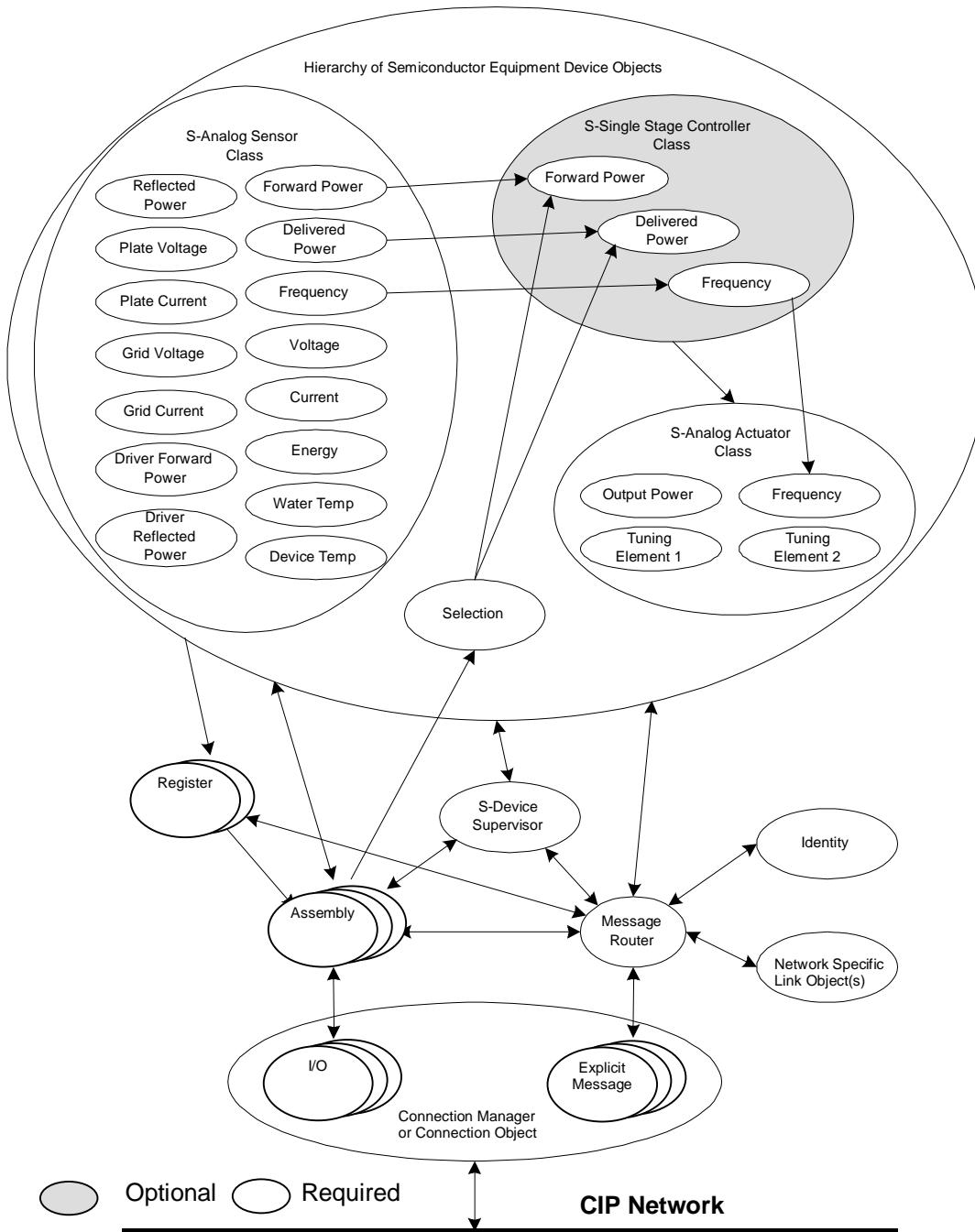
**RF Power Generator Device, Type: 20<sub>Hex</sub>**

**Table 6-39.3 S-Single Stage Controller Object Subclasses**

<b>Object Instance ID</b>	<b>Subclass Name</b>	<b>Subclass ID (Attribute 99 Value)</b>	<b>Required</b>	<b>Function</b>	<b>Restrictions</b>
1	RF Generator	03	Optional	Forward Power Regulation	Data Units = Watts or milliWatts
2	RF Generator	03	Optional	Delivered Power Regulation	Data Units = Watts
3	Frequency Control	04	Optional	Frequency Regulation	Data Units = Hertz

RF Power Generator Device, Type: 20<sub>Hex</sub>

Figure 6-39.4 Object Model for the RF Power Generator Device



### 6-39.3 How Objects Affect Behavior

**Table 6-39.5 Object Affect on Behavior**

Object	Effect on behavior
Identity	Supports the Reset service. Upon receipt of a <i>Reset Service Request</i> of any <i>Type</i> , the Identity Object sends a <i>Reset Service Request</i> to the S-Device Supervisor.
CIP Common Required	See Section 6-2.2 for details.
Assembly	Defines input/output and configuration data format
S-Device Supervisor	Supports the Stop, Start, Reset, Abort, Recover and Perform_Diagnostic services for ALL Application Objects in the device and consolidates the Exception Conditions and Application Objects' Status.  This object behaves differently from the Identity Object in that the S-Device Supervisor object provides a single point of access to the Application Objects only; it does not effect the DeviceNet specific objects (i.e., Identity, DeviceNet, Connection, etc.).
S-Analog Sensor	Provides sensed measurements and feeds process variables to the S-Single Stage Controllers
S-Single Stage Controller	The active instance feeds the control variable to the S-Analog Actuator thereby regulating the output as specified
S-Analog Actuator	Operates the Power Generator output drive circuitry
Selection	Activates an instance of the S-Single Stage Controller class and feeds it the Setpoint variable
Register	Instance 1 - No effect.  Instance 2 - The transition of any interlock bit from 0 to 1 causes an internally generated Stop Service Request to the S-Device Supervisor object instance.

### 6-39.4 Defining Object Interfaces

**Table 6-39.6 Object Interfaces**

Object	Interface
CIP Common Required	See Section 6-2.3 for details.
Assembly	I/O Connection or Message Router
S-Device Supervisor	Assembly or Message Router
S-Analog Sensor	Assembly or Message Router
S-Analog Actuator	Assembly or Message Router
S-Single Stage Controller	Assembly or Message Router
Selection	Assembly or Message Router
Register	Assembly or Message Router

## 6-39.5 I/O Assembly Instances

The following table identifies the I/O assembly instances. The manufacturer must specify which Assembly instances are supported by the device.

Any element not supported must be set to zero (0).

**Table 6-39.7 I/O Assembly Instances**

Instance	Required	Type	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	Y (Default)	Output	0	Forward Power Out 12 (low byte)							
			1	0	0	0	0	Power Out 12 (high nibble)			
			2	0	0	0	0	0	0	0	0
			3	0	0	0	0	0	0	0	0
			4	0	0	0	0	0	0	0	On
2	Y (Default)	Input	0	Forward Power In 12 (low byte)							
			1	0	0	0	0	Forward Power In 12 (high nibble)			
			2	Reflected Power In 12 (low byte)							
			3	0	0	0	0	Reflected Voltage In 12 (high nibble)			
			4	0	0	0	0	0	0	0	0
			5	0	0	0	0	0	0	0	0
			6	0	0	0	0	0	0	0	0
			7	0	0	0	0	0	0	0	0
			8	Op Status							
3	Y	Output	0	0	0	0	0	0	0	0	On
4	N	Output	0 - 3	Forward Power Limit							
			4 - 7	Delivered Power Limit							
			8 - 11	Frequency Limit Low							
			12 - 15	Frequency Limit High							
5	N	Output	0 - 1	Regulation Select							
			2 - 5	Setpoint							
			6 - 7	Power Ramp							
			8	0	0	0	0	0	0	0	On
6	N	Output	0 - 1	Regulation Select							
			2 - 5	Setpoint							
			6	0	0	0	0	0	0	0	On
7	N	Output	0 - 3	Forward Power Out							
			4	0	0	0	0	0	0	0	On
8	N	Output	0 - 3	Forward Power Out							
			4 - 5	Power Ramp							
			6	0	0	0	0	0	0	0	On
9	N	Output	0 - 3	Delivered Power Out							
			4	0	0	0	0	0	0	0	On

RF Power Generator Device, Type: 20<sub>Hex</sub>

Instance	Required	Type	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
10	N	Output	0 - 3	Delivered Power Out							
			4 - 5	Power Ramp							
			6	0	0	0	0	0	0	0	On
11	N	Input	0	Ex Status							
			1 - 4	Forward Power In							
			5 - 8	Voltage In							
			9 - 12	Current In							
12	N	Input	0	Ex Status							
			1 - 4	Energy In							
13	N	Input	0	Ex Status							
			1 - 4	Forward Power In							
			5 - 8	Voltage In							
			9 - 12	Current In							
			13 - 16	Energy In							
14	N	Input	0	Ex Status							
			1 - 4	Forward Power In							
			5 - 8	Reflected Power In							
			9 - 12	Voltage In							
			13 - 16	Current In							
15	N	Input	0	Ex Status							
			1 - 4	Delivered Power In							
16	N	Input	0	Ex Status							
			1 - 2	W Temp							
			3 - 4	D Temp							
17	Y	Input	0	Ex Status							
18	Y	Input	0	Exception Detail Alarm 0 (size, common = 2)							
			1	Exception Detail Alarm 1 (common 0)							
			2	Exception Detail Alarm 2 (common 1)							
			3	Exception Detail Alarm 3 (size, device = 2)							
			4	Exception Detail Alarm 4 (device 0)							
			5	Exception Detail Alarm 5 (device 1)							
			6	Exception Detail Alarm 6 (size, manufacturer = 1)							
			7	Exception Detail Alarm 7 (manufacturer 0)							
19	Y	Input	0	Exception Detail Warning 0 (size, common = 2)							
			1	Exception Detail Warning 1 (common 0)							
			2	Exception Detail Warning 2 (common 1)							
			3	Exception Detail Warning 3 (size, device = 2)							
			4	Exception Detail Warning 4 (device 0)							
			5	Exception Detail Warning 5 (device 1)							
			6	Exception Detail Warning 6 (size, manufacturer = 1)							
			7	Exception Detail Warning 7 (manufacturer 0)							

RF Power Generator Device, Type: 20<sub>Hex</sub>

Instance	Required	Type	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0					
20	N	Input	0	Ex Status												
			1	Exception Detail Alarm 0 (size, common = 2)												
			2	Exception Detail Alarm 1 (common 0)												
			3	Exception Detail Alarm 2 (common 1)												
			4	Exception Detail Alarm 3 (size, device = 2)												
			5	Exception Detail Alarm 4 (device 0)												
			6	Exception Detail Alarm 5 (device 1)												
			7	Exception Detail Alarm 6 (size, manufacturer = 1)												
			8	Exception Detail Alarm 7 (manufacturer 0)												
			9	Exception Detail Warning 0 (size, common = 2)												
			10	Exception Detail Warning 1 (common 0)												
			11	Exception Detail Warning 2 (common 1)												
			12	Exception Detail Warning 3 (size, device = 2)												
			13	Exception Detail Warning 4 (device 0)												
			14	Exception Detail Warning 5 (device 1)												
			15	Exception Detail Warning 6 (size, manufacturer = 1)												
			16	Exception Detail Warning 7 (manufacturer 0)												
21	N	Input	0	Op Status												
22	N	Input	0	Interlock Status (low byte)												
			1	0	0		Interlock Status (high 6 bits)									
23	N	Input	0	Ex Status												
			1	Op Status												
			2	Interlock Status (low byte)												
			3	0	0		Interlock Status (high 6 bits)									

**6-39.5.1 I/O Assembly Instance Component Formats**

All of the elemental components listed above follow standard CIP Data Encoding as specified in Appendix C. The encoding order for all data is Low Byte first. All members are LSB aligned and end on Byte boundaries. That is, upper bits are stuffed with 0 as needed to fill to the next byte.

**6-39.5.2 I/O Assembly Instance Component Mapping**

Each of the *S-Analog Sensor*, *S-Analog Actuator* and *S-Single Stage Controller* object definitions specifies a behavior that modifies the *Data Type* of certain attributes based upon the first valid I/O connection established to an Assembly Object instance. See these object definitions for more detail.

RF Power Generator Device, Type: 20<sub>Hex</sub>

The following table indicates the I/O assembly Data attribute mapping for this device.

Table 6-39.8 I/O Assembly Data Mapping

Data Component	Object Name	Class ID	Instance ID	Attribute Name	Attribute ID	Data Type
<b>Forward Power Out 12</b>	S-Single Stage Controller	33 <sub>hex</sub>	1	Setpoint	6	UINT
<b>Forward Power In 12</b>	S-Analog Sensor	31 <sub>hex</sub>	1	Value	6	UINT
<b>Reflected Power In 12</b>	S-Analog Sensor	31 <sub>hex</sub>	2	Value	6	UINT
<b>Power Ramp</b>	S-Single Stage Controller	33 <sub>hex</sub>	1	Ramp Rate Increment	96	UINT
<b>On</b>	S-Device Supervisor	30 <sub>hex</sub>	1	Output Power Enable	96	BOOL
<b>Op Status</b>	Register	07 <sub>hex</sub>	1	Data	4	ARRAY of 8 BITS
<b>Forward Power Limit</b>	S-Single Stage Controller	33 <sub>hex</sub>	1	Output Limit	86	DINT
<b>Delivered Power Limit</b>	S-Single Stage Controller	33 <sub>hex</sub>	2	Output Limit	86	DINT
<b>Frequency Limit Low</b>	S-Single Stage Controller	33 <sub>hex</sub>	3	Frequency Limit Low	95	DINT
<b>Frequency Limit High</b>	S-Single Stage Controller	33 <sub>hex</sub>	3	Frequency Limit High	96	DINT
<b>Filament On/Off</b>	S-Device Supervisor	30 <sub>hex</sub>	1	Filament Power Enable	95	BOOL
<b>Regulation Select</b>	Selection	2E <sub>hex</sub>	1	destination_used	14	UINT
<b>Setpoint</b>	Selection	2E <sub>hex</sub>	1	input_data_value	15	DINT
<b>Forward Power Out</b>	S-Single Stage Controller	33 <sub>hex</sub>	1	Setpoint	6	DINT
<b>Forward Power In</b>	S-Analog Sensor	31 <sub>hex</sub>	1	Value	6	DINT
<b>Reflected Power In</b>	S-Analog Sensor	31 <sub>hex</sub>	2	Value	6	DINT
<b>Voltage In</b>	S-Analog Sensor	31 <sub>hex</sub>	4	Value	6	DINT
<b>Current In</b>	S-Analog Sensor	31 <sub>hex</sub>	5	Value	6	DINT
<b>Energy In</b>	S-Analog Sensor	31 <sub>hex</sub>	7	Value	6	DINT
<b>Delivered Power Out</b>	S-Single Stage Controller	33 <sub>hex</sub>	2	Setpoint	6	DINT
<b>Delivered Power In</b>	S-Analog Sensor	31 <sub>hex</sub>	3	Value	6	DINT
<b>W Temp</b>	S-Analog Sensor	31 <sub>hex</sub>	8	Value	6	INT
<b>D Temp</b>	S-Analog Sensor	31 <sub>hex</sub>	9	Value	6	INT
<b>Interlock</b>	Register	07 <sub>hex</sub>	2	Data	4	ARRAY of 16 BITS
<b>Ex Status</b>	S-Device Supervisor	30 <sub>hex</sub>	1	Exception Status	12	BYTE
<b>Ex Alarm</b>	S-Device Supervisor	30 <sub>hex</sub>	1	Exception Detail Alarm	13	STRUCT
<b>Ex Warn</b>	S-Device Supervisor	30 <sub>hex</sub>	1	Exception Detail Warning	14	STRUCT

## 6-39.6 Object Limitations and Specific Definitions

This section describes limitations and specific definitions applicable to the listed objects of this device profile when these objects are applied in this type of device.

### 6-39.6.1 S-Device Supervisor

The following attribute limitations apply:

**Table 6-39.9 Device Type Attribute Limitation**

Attribute	Requirements
Device Type	Shall be set to "RFG" for RF Power Generator Device

The following table specifies the data attribute bit mapping for the **Device Exception Detail** bytes for this device. For more descriptive information, see the definition of the S-Device Supervisor Object Class.

Any Exception Bit not supported shall default to 0. Note that this profile allows for only one byte of manufacturer specific exception detail.

**Table 6-39.10 Data Attribute Bit Mapping**

Data Component	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
<b>Device Exception Detail Size</b>	0	0	0	0	0	1	0	0
<b>Device Exception Detail Byte 0</b>	Magnetron Assembly Fault	Power Detector Fault	Contactor Closed	Fan Failure	Cooling Failure	Device Temp High	Interlock Open	Control Circuit Fault
<b>Byte 1</b>	Input Power V Low	Input Power V High *	Output I Low	Output V Low	Output V High	Reflecte d P High	Output P High	Regulation
<b>Byte 2</b>	IGBT Driver	Xfmr I High	Magnet V High	Bus V High	Bus V Low	Bus Shorted	DC I High	DC V Error
<b>Byte 3</b>	Reserved 0	Reserved 0	End of Target Life *	Dew Condensation	Non-Zero Idle Output I	Non-Zero Idle Output V	Anode I High	Anode V High
<b>Manufacturer Exception Detail Size</b>	0	0	0	0	0	0	0	1
<b>Manufacturer Exception Detail Byte 0</b>	8 Bits defined by Manufacturer							

\* Only used in the Warning Exception Detail, this bit is always = 0 in the Alarm Exception Detail.

### 6-39.6.2 S-Analog Sensor Object Instances

Table 6-39.11 Object Instances

Object Instance ID	Function	Required	Limitations and Specific Definitions
1	Forward Power	Required	Supported Data Type and Data Units pairs: DINT / Watts; DINT / milliWatts UINT / % *
2	Reflected Power	Required	Supported Data Type and Data Units pairs: DINT / Watts (rms); DINT / milliWatts (rms) UINT / % *
3	Delivered Power	Optional	Supported Data Type and Data Units pairs: DINT / Watts (rms); DINT / milliWatts (rms) UINT / % *
4	Output Voltage	Optional	Supported Data Type and Data Units pairs: DINT / Volts (rms); UINT / % *
5	Output Current	Optional	Supported Data Type and Data Units pairs: DINT / milliAmps (rms); UINT / % *
6	Output Frequency	Required	DINT / Hertz
7	Delivered Energy since Output Power was last turned On	Optional	DINT / Joule
8	Cooling Water Temp	Optional	INT / °C Resolution = 1 Degree
9	Device Temp	Optional	INT / °C Resolution = 1 Degree
10	Plate Voltage	Optional	UINT / Volts
11	Plate Current	Optional	UINT / milliAmps
12	Grid Voltage	Optional	UINT / Volts
13	Grid Current	Optional	UINT / milliAmps
14	Driver Forward Power	Optional	UINT / Watts
15	Driver Reflected Power	Optional	UINT / Watts

\* Range = 0 - 4095 corresponding to 0 - 100%

### 6-39.6.3 S-Analog Actuator Object Instances

**Table 6-39.12 Object Instances**

Object Instance ID	Function	Required	Limitations and Specific Definitions
1	Output Power	Required	Supported Data Type and Data Units pairs: DINT / Watts; DINT / milliWatts UINT / % *
2	Output Frequency	Required	Supported Data Type and Data Units pairs: DINT / Hertz; UINT / % *
3	Variable Tuning Element 1	Optional	USINT / % of Max
4	Variable Tuning Element 2	Optional	USINT / % of Max

\* Range = 0 - 4095 corresponding to 0 - 100%

### 6-39.6.3.1 S-Single Stage Controller Object Instances

**Table 6-39.13 Object Instances**

Object Instance ID	Function	Required	Limitations and Specific Definitions
1	Output Forward Power Regulation	Optional	Supported Data Type and Data Units pairs: DINT / Watts; DINT / milliWatts UINT / % *
2	Output Delivered Power Regulation	Optional	Supported Data Type and Data Units pairs: DINT / Watts; DINT / milliWatts UINT / % *
3	Output Frequency Regulation	Optional	Supported Data Type and Data Units pairs: DINT / Hertz; UINT / % *

\* Range = 0 - 4095 corresponding to 0 - 100%

### 6-39.6.3.2 Behavior

Of instances 1 and 2, only one is active at any given time. Which instance is active is determined by the value of the *destination\_used* attribute of the Selection object instance. In typical operation, the *destination\_used* attribute is mapped to, and hence modified by, the Regulation Select member of an Output Assembly Instance, which in turn, is referenced by the *Consume Path* of the first valid I/O connection established by the device.

#### 6-39.6.4 Selection Object, Instance 1

**Table 6-39.14 Object Instances**

Attribute	Limitation (assigned values are shown in hex format)
Max_destination	02
Number_of_destinations	02
Destination_list	06 / 20 / 33 / 24 / 01 / 30 / 06 — S-Single Stage Controller, Instance 1, Setpoint 06 / 20 / 33 / 24 / 02 / 30 / 06 — S-Single Stage Controller, Instance 2, Setpoint Access Rule = Get Only
Max_sources	00
Number_of_sources	00
Source_list	Not Supported
Source_used	00
Algorithm_used	04 - Programmable Data Flow. Access Rule = Get Only
Object_source_list	00 Access Rule = Get Only
Destination_used	Range = 0 - 2

#### 6-39.6.5 Register Object

##### 6-39.6.5.1 Register Object, Instance 1 Operational Status

**Table 6-39.15 Object Instances**

Bit	Description
0	Power On Status (1=on / 0=off)
1	Setpoint Status (1=at setpoint / 0=not)
2	Temp Status (1=OK / 0=overtemp)
3	Reserved (0)
4	Interlock Status (1=OK / 0=open)
5	Reserved for Future Use (0)
6	Reserved for Future Use (0)
7	Reserved for Future Use (0)

**6-39.6.5.2 Register Object, Instance 2 Interlock Status****Table 6-39.16 Object Instances**

<b>Bit</b>	<b>Description</b>
0	Device Not Ready
1	Communications Fault
2	Device State Fault
3	Device Not Configured
4	External
5	Head Cable
6	Head Cover
7	High-Voltage Cable
8	Bias Supply
9	DC Bus Voltage
10	Fan Failure
11	Magnet Temp
12	Power Monitor
13	Cover
14	Reserved for Future Use (0)
15	Reserved for Future Use (0)

1 = Interlocked / 0 = Not Interlocked

**6-39.6.5.3 Behavior**

The transition of any interlock bit from 0 to 1 shall cause an internally generated Stop Service Request to the S-Device Supervisor object instance thereby causing a transition to the IDLE state and disabling the output power of the device. Also, if any interlock bit has a value of 1, the S-Device Supervisor object instance shall not transition to the EXECUTING State: (a) a Start Service Request shall return a Device\_State\_Conflict error, and (b) a Receipt of First Valid I/O Data event shall not cause a state transition.

**6-39.7 Defining Device Configuration**

Public access to the S-Device Supervisor and the application objects by the Message Router must be supported for configuration of this device type.

## 6-40 Fluid Flow Controller Device

### Device Type: 24 hex

A Fluid Flow Controller (FFC) is a device that measures and controls the flow rate of a fluid (liquid or gas). It contains three principle components: a flow rate sensor which can be a virtual sensor based on other sensors, for example pressure; a fluid flow metering valve which can be actuated by one of a variety of actuator types, including, for example, solenoid or a stepper motor; and, a controller which closes the loop by receiving a setpoint and driving the actuator such that the fluid flow rate is controlled to the setpoint.

### 6-40.1 Object Model

The Object Model in Figure 6-40.2 represents a Fluid Flow Controller Device. The table below indicates:

- the object classes present in this device
- whether or not the class is required
- the number of instances present in each class

**Table 6-40.1 Objects Present in a Fluid Flow Controller Device**

Object Class	Optional/Required	# of Instances
Identity	Required	1
Message Router	Required	1
DeviceNet	Required	1
Connection	Required	at least 1 I/O and 1 Explicit
Assembly	Required	at least 1 Input and 1 Output
S-Device Supervisor	Required	1
S-Sensor Calibration * or S-Gas Calibration *	Optional	0 or More
S-Analog Sensor	Required	1 or More
S-Analog Actuator	Conditional **	1
S-Single Stage Controller	Conditional **	1

\* For FFC devices that flow **Liquid**, the **S-Sensor Calibration object** is used.

For FFC devices that flow **Gas**, the **S-Gas Calibration object** is used.

\*\* Required for a Fluid Flow Controller, a device that contains a Valve and a Controller. Not supported in a Fluid Flow Meter Device.

## **6-40.2 Class Subclasses**

Each class level subclass defines a unique meaning for an overlapping range of class attribute IDs and/or class service IDs. The range for subclass definitions begins at ID 96 and numbers downward for attributes, and ID 63<sub>hex</sub> and numbers downward for services. The subclass for a given class is identified by the value of its Subclass class attribute. There are no class level subclasses specified for this device.

## **6-40.3 Instance Subclasses**

Each instance level subclass defines a unique meaning for an overlapping range of instance attribute IDs and/or instance service IDs. The range for subclass definitions begins at ID 96 and numbers downward for attributes, and ID 63<sub>hex</sub> and numbers downward for services. The subclass for a given instance is identified by the value of its Subclass instance attribute.

The only instance level subclass specified for this device is:

- S-Analog Sensor Object, Instance Identifier 01 -- Subclass 01, Flow Diagnostics

## **6-40.4 Instance Identifiers**

Additional objects are restricted for the purposes of reserving instance number ranges for future revisions of this device profile. As permitted by CIP, vendor-specific objects (object classes in the range 100-199 or 768-1279) may be added without instance number restrictions. However, open objects (i.e., all objects defined by the CIP Common Specification, but not included in "CIP Common Specification, Table 6-2.2, Minimum Objects Required for All Devices") may only be added (as vendor-specific additions) using instance numbers in the range 100-199 or 768-1279, the vendor-specific open object instance number range for this device type. This allows object extensions to this device profile specification in future revisions (i.e., added open objects defined with instance identifiers < 100).

## Fluid Flow Controller Device, Type: 24 Hex

Figure 6-40.2 Object Model for the FFC Device

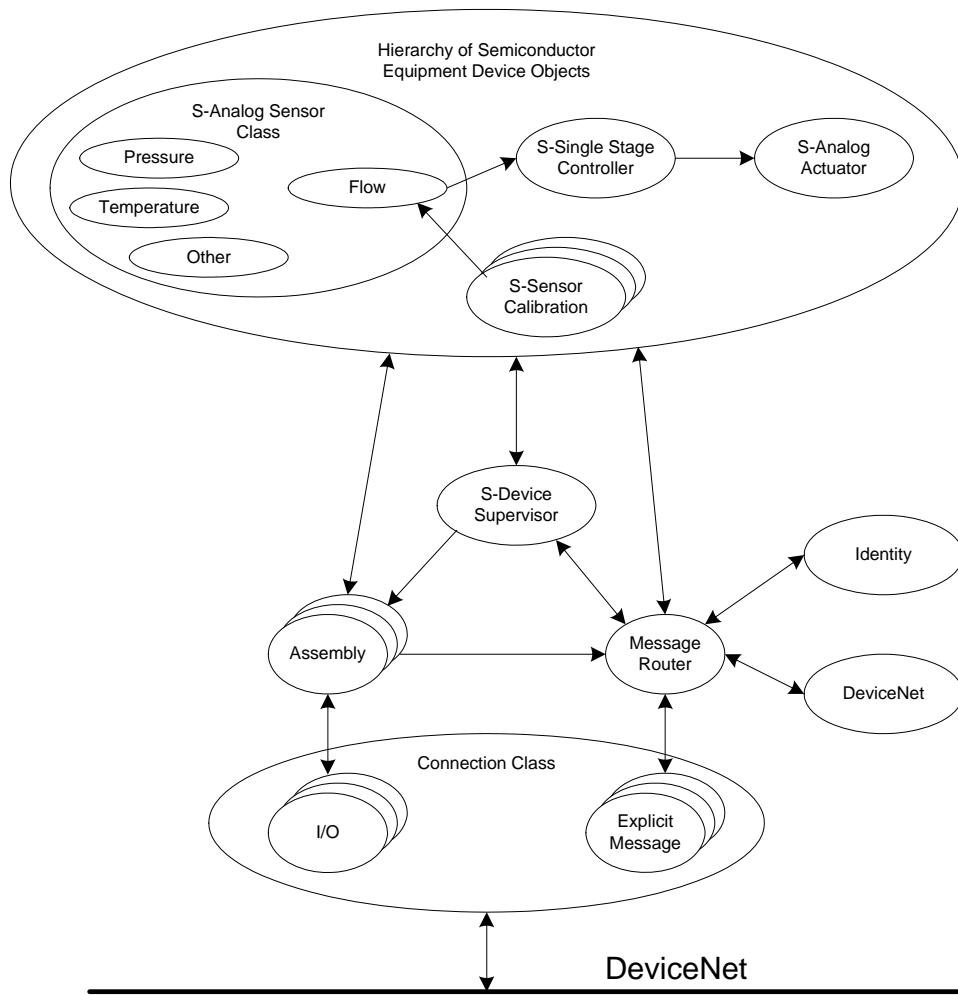


Table 6-40.3 S-Analog Sensor Instances

S-Analog Sensor	Name	Description
Instance 1 *	Flow	Virtual sensor that calculates the fluid flow rate based on various sensed parameters, for example differential pressure
Instance 2 (optional)	Pressure	The measured fluid pressure
Instance 3 (optional)	Temperature	The measured fluid temperature
Instance 4 (optional)	Density	The measured fluid density

\* Subclass 01, Flow Diagnostics

## 6-40.5 How Objects Affect Behavior

**Table 6-40.4 Object Affect on Behavior**

Object	Effect on behavior
Identity	Supports the Reset service. Upon receipt of a <i>Reset Service Request</i> of any <i>Type</i> , the Identity Object sends a <i>Reset Service Request</i> to the S-Device Supervisor.
Message Router	No effect
DeviceNet	Configures port attributes (node address, data rate, and BOI)
Connection Class	Contains the number of logical ports into or out of the device
Assembly	Defines input/output and configuration data format
S-Device Supervisor	Supports the Stop, Start, Reset, Abort, Recover and Perform_Diagnostic services for ALL Application Objects in the device and consolidates the Exception Conditions and Application Objects' Status.  This object behaves differently from the Identity Object in that the S-Device Supervisor object provides a single point of access to the Application Objects only; it does not effect the DeviceNet specific objects (i.e., Identity, DeviceNet, Connection, etc.).
S-Sensor Calibration	Modifies the correction algorithm of the related S-Analog Sensor object which includes the selection mechanism to enable an S-Sensor Calibration object instance.
S-Analog Sensor	A virtual sensor determines the measured fluid flow rate or other parameter. The flow sensor feeds the process variable to the S-Single Stage Controller object
S-Single Stage Controller	Feeds the control variable to the S-Analog Actuator object
S-Analog Actuator	Operates the Flow Control Valve of the device

## 6-40.6 Defining Object Interfaces

**Table 6-40.5 Object Interfaces**

Object	Interface
Identity	Message Router
Message Router	Explicit Messaging Connection Instance
DeviceNet	Message Router
Connection Class	Message Router
Assembly	I/O Connection or Message Router
S-Device Supervisor	Assembly or Message Router
S-Sensor Calibration	Message Router
S-Analog Sensor	Assembly or Message Router
S-Single Stage Controller	Assembly or Message Router
S-Analog Actuator	Assembly or Message Router

## 6-40.7 I/O Assembly Instances

The following table identifies the I/O assembly instances supported by the FFC. As specified by the Assembly Object definition, the range of instances up to 99 are reserved for future specification revision to this profile.

**Fluid Flow Controller Device, Type: 24 Hex****Table 6-40.6 FFC I/O Assembly Instances**

<b>Number</b>	<b>Required</b>	<b>Type</b>	<b>Name</b>
1	Y	Input	Flow
2	Y (default)	Input	Status and Flow
3	N	Input	Status, Flow and Pressure
4	N	Input	Status, Flow and Temperature
5	N	Input	Status, Flow, Pressure and Temperature
6	N	Input	Status, Flow, Pressure and Valve
7	N	Input	Status, Flow, Pressure, Temperature and Valve
8	N	Input	Status, Flow, Setpoint and Valve
9	N	Input	Status, Flow, Setpoint, Override and Valve
10	C* (default)	Output	Setpoint
11	C*	Output	Override and Setpoint
12	N	Input	FP-Flow
13	N	Input	Status and FP-Flow
14	N	Input	Status, FP-Flow and FP-Pressure
15	N	Input	Status, FP-Flow and FP-Temperature
16	N	Input	Status, FP-Flow, FP-Pressure and FP-Temperature
17	N	Input	Status, FP-Flow, FP-Pressure and FP-Valve
18	N	Input	Status, FP-Flow, FP-Pressure, FP-Temperature and FP-Valve
19	N	Input	Status, FP-Flow, FP-Setpoint and FP-Valve
20	N	Input	Status, FP-Flow, FP-Setpoint, FP-Override and FP-Valve
21	N	Output	FP-Setpoint
22	N	Output	Override and FP-Setpoint
23	N	Input	Status, Flow, Density
24	N	Input	Status, Flow, Density, Temperature
25	N	Input	Status, Flow, Pressure, Temperature and Density
26	N	Input	Status, Flow, Pressure, Temperature, Density and Valve
27	N	Input	Status, FP-Flow, FP-Density
28	N	Input	Status, FP-Flow, FP-Density, FP-Temperature
29	N	Input	Status, FP-Flow, FP-Pressure, FP-Temperature and FP-Density
30	N	Input	Status, FP-Flow, FP-Pressure, FP-Temperature, FP-Density and FP-Valve

\* Required for a Fluid Flow Controller, a device that contains a Valve and a Controller. Not supported in a Fluid Flow Meter Device.

## **6-40.8 I/O Assembly Object Instance Data Attribute Encoding**

The manufacturer of a Fluid Flow Controller Device must specify which Assembly instances are supported by the device.

The I/O Assembly DATA attribute is formatted as a structure of data elements with the first element ordered first. All data components are encoded with the Low-ordered Byte first; that is, the Least Significant Byte is in the Low-ordered Index (Address) i.e., Little Endian.

## 6-40.9 Mapping I/O Assembly Data Attribute Components

Each of the *S-Analog Sensor*, *S-Analog Actuator* and *S-Single Stage Controller* object definitions specifies a behavior that modifies the *Data Type* of certain attributes based upon the first valid I/O connection established to an Assembly Object instance. In order to maintain consistency, this device type is limited to only allow connections to either INT or REAL based Assembly instances. Once a valid connection is established, attempts to configure connections to a different type of Assembly instance will return an error.

The following table indicates the I/O assembly Data attribute mapping for this FFC device.

**Table 6-40.7 I/O Assembly Data Mapping**

<b>Data Component</b>	<b>Class</b>		<b>Instance Number</b>	<b>Attribute</b>		
	<b>Name</b>	<b>Name</b>		<b>Name</b>	<b>Number</b>	<b>Type</b>
Flow	S-Analog Sensor	31 <sub>hex</sub>	1	Value	6	INT
Pressure	S-Analog Sensor	31 <sub>hex</sub>	2	Value	6	INT
Temperature	S-Analog Sensor	31 <sub>hex</sub>	3	Value	6	INT
Density	S-Analog Sensor	31 <sub>hex</sub>	4	Value	6	INT
Valve	S-Analog Actuator	32 <sub>hex</sub>	1	Value	6	INT
Override	S-Analog Actuator	32 <sub>hex</sub>	1	Override	5	USINT
Setpoint	S-Single Stage Controller	33 <sub>hex</sub>	1	Setpoint	6	INT
Status	S-Device Supervisor	30 <sub>hex</sub>	1	Exception Status	12	BYTE
FP-Flow	S-Analog Sensor	31 <sub>hex</sub>	1	Value	6	REAL
FP-Pressure	S-Analog Sensor	31 <sub>hex</sub>	2	Value	6	REAL
FP-Temperature	S-Analog Sensor	31 <sub>hex</sub>	3	Value	6	REAL
FP-Density	S-Analog Sensor	31 <sub>hex</sub>	4	Value	6	REAL
FP-Valve	S-Analog Actuator	32 <sub>hex</sub>	1	Value	6	REAL
FP-Setpoint	S-Single Stage Controller	33 <sub>hex</sub>	1	Setpoint	6	REAL

## 6-40.10 Object Limitations and Specific Definitions

This section describes limitations and specific definitions applicable to the listed objects of this device profile when these objects are applied in this type of device.

### 6-40.10.1 S-Device Supervisor Object Instance

**Table 6-40.8 Limitations**

Attribute	Limitation
Device Type	Supported Values: "FFC" = Fluid Flow Controller device (Liquid Type) "FFM" = Fluid Flow Meter device (Liquid Type) "FCG" = Fluid Flow Controller device (Gas Type) "FMG" = Fluid Flow Meter device (Gas Type)

### 6-40.10.2 Specific Definition

The following table specifies the data attribute bit mapping for the **Device Exception Detail** bytes for this FFC device. For more descriptive information, see the definition of the S-Device Supervisor Object Class. Noted, for each entry, is the Object from which the Status byte/bit is mapped. See the object specification for the detailed bit mapping.

Any Exception Bit not supported must default to 0. Note that this profile allows for only one byte of manufacturer specific exception detail.

**Table 6-40.9 Bit Mapping for Device Exception Detail**

Data Component	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
<b>FFC Device Exception Detail Size</b>	0	0	0	0	0	0	1	1
<b>FFC Device Exception Detail [0]</b>	Pressure High S-Analog Sensor 2	Pressure Low S-Analog Sensor 2	Valve High S-Analog Actuator	Valve Low S-Analog Actuator	Flow Control S-Single Stage Controller	Flow High S-Analog Sensor 1	Flow Low S-Analog Sensor 1	Reading Valid * S-Analog Sensor 1
<b>FFC Device Exception Detail [1]</b>	0	0	0	0	Density High S-Analog Sensor 4	Density Low S-Analog Sensor 4	Temp High S-Analog Sensor 3	Temp Low S-Analog Sensor 3
<b>FFC Device Exception Detail [2]</b>	0	0	0	0	0	0	0	0
<b>Manufacturer Exception Detail Size</b>	0	0	0	0	0	0	0	1
<b>Manufacturer Exception Detail</b>	8 Bits defined by Manufacturer							

\* Only used in the Warning Exception Detail, this bit is always = 0 in the Alarm Exception Detail

### 6-40.10.3 S-Analog Sensor Object

**Table 6-40.10 Limitations Instance 1**

Attribute	Limitation	Requirement	Default
Data Type	Supported Values = {INT; REAL}	Supported Values = {INT}	INT
Data Units	Supported Values = {Counts, % & Units of the Flow Group} (see Appendix K)	Supported Values = {counts}	Counts *
Subclass		01 = Flow Diagnostics	

**Table 6-40.11 Limitations Instances 2**

Attribute	Limitation	Requirement	Default
Data Type	Supported Values = {INT; REAL}	Supported Values = {INT}	INT
Data Units	Supported Values = {Counts, % & Units of the Pressure Group} (see Appendix K)	Supported Values = {counts}	Counts *

**Table 6-40.12 Limitations Instance 3**

Attribute	Limitation	Requirement	Default
Data Type	Supported Values = {INT; REAL}	Supported Values = {INT}	INT
Data Units	Supported Values = {Counts, % & Units of the Temperature Group} (see Appendix K)	Supported Values = {counts}	Counts *

**Table 6-40.13 Limitations Instance 4**

Attribute	Limitation	Requirement	Default
Data Type	Supported Values = {INT; REAL}	Supported Values = {INT}	INT
Data Units	Supported Values = {Counts, % & Units of the Density Group} (see Appendix K)	Supported Values = {counts}	Counts *

\* Counts are interpreted as a "Percent of Full Scale" where a Count Value of 24576 (6000 hexadecimal) corresponds to 100%. This allows a maximum reading of 133.33%.

### 6-40.10.4 S-Analog Actuator Object

**Table 6-40.14 Limitations**

Attribute	Limitation	Requirement	Default
Data Type	Supported Values = {INT; REAL}	Supported Values = {INT}	INT
Data Units	Supported Values = {Counts, %, Voltage, Current & Units of the Flow Group} (see Appendix K)	Supported Values = {counts}	Counts *

\* Counts are interpreted as a "Percent of Full Scale" where a Count Value of 24576 (6000 hexadecimal) corresponds to 100%. This allows a maximum reading of 133.33%.

### 6-40.10.5 S-Single Stage Controller Object

**Table 6-40.15 Limitations**

Attribute	Limitation	Requirement	Default
Data Type	Supported Values = {INT; REAL}	Supported Values = {INT}	INT
Data Units	Supported Values = {Counts, %, & Units of the Flow Group} (see Appendix K)	Supported Values = {counts}	Counts *
Process Variable	Not accessible over the network. The <i>Process Variable</i> input to this object instance is the value of the S-Analog Sensor object instance <i>Value</i> attribute.	N.A.	N.A.
CV Data Type	Not supported	N.A.	N.A.
Control Variable	Not accessible over the network, The <i>Control Variable</i> output from this object is the value of the S-Analog Actuator object instance <i>Value</i> attribute.	N.A.	N.A.

\* Counts are interpreted as a "Percent of Full Scale" where a Count Value of 24576 (6000 hexadecimal) corresponds to 100%. This allows a maximum reading of 133.33%.

The **Data Type** and **Data Units** of the S-Single Stage Controller object shall remain consistent with those of the S-Analog Sensor object Instance 1 (Flow). Any change to the value of the Data Type or Data Units attribute of either of these two object instances will cause the device to self-modify the value of the corresponding attribute of the other object.

### 6-40.11 Defining Device Configuration

Public access to the S-Device Supervisor, S-Analog Sensor, S-Analog Actuator, S-Single Stage Controller, and S-Sensor Calibration Objects by the Message Router must be supported for configuration of this device type.

## 6-41 CIP Motion Drive

**Device Type: 25<sub>hex</sub>**

### 6-41.1 Introduction

A CIP Motion™ Drive device controls the motion of one or more motion axes, typically consisting of a rotary or linear motor actuator and one or more optional feedback devices. Each associated motor is driven by an associated drive power structure that is also part of the device. Motor control functionality supported by this profile can be applied to “open loop” Variable Frequency Drives or “closed loop” Vector Controlled Servo Drives. In either case, axis motion is controlled via a command reference that can be configured for position control, velocity control, acceleration control, or current/torque control. This profile also supports the use of additional motion axis feedback channels as master feedback sources.

### 6-41.2 Object Model

The object model in Figure 6-41.1 represents a CIP Motion Drive device. The table below indicates:

- the object classes present in this device
- whether or not the class is required
- the number of instances present in each class

Chapter 5, Object Library, provides more details about these objects.

**Table 6-41.1 Object Present in a CIP Motion Drive Device**

Object Class	Optional or Required	# of Instances
CIP Required Objects	Required	See Chapter 6-2.1
Time Sync Object	Optional	1
Motion Axis Object	Required	1 per Axis

### 6-41.2.1 Object Description

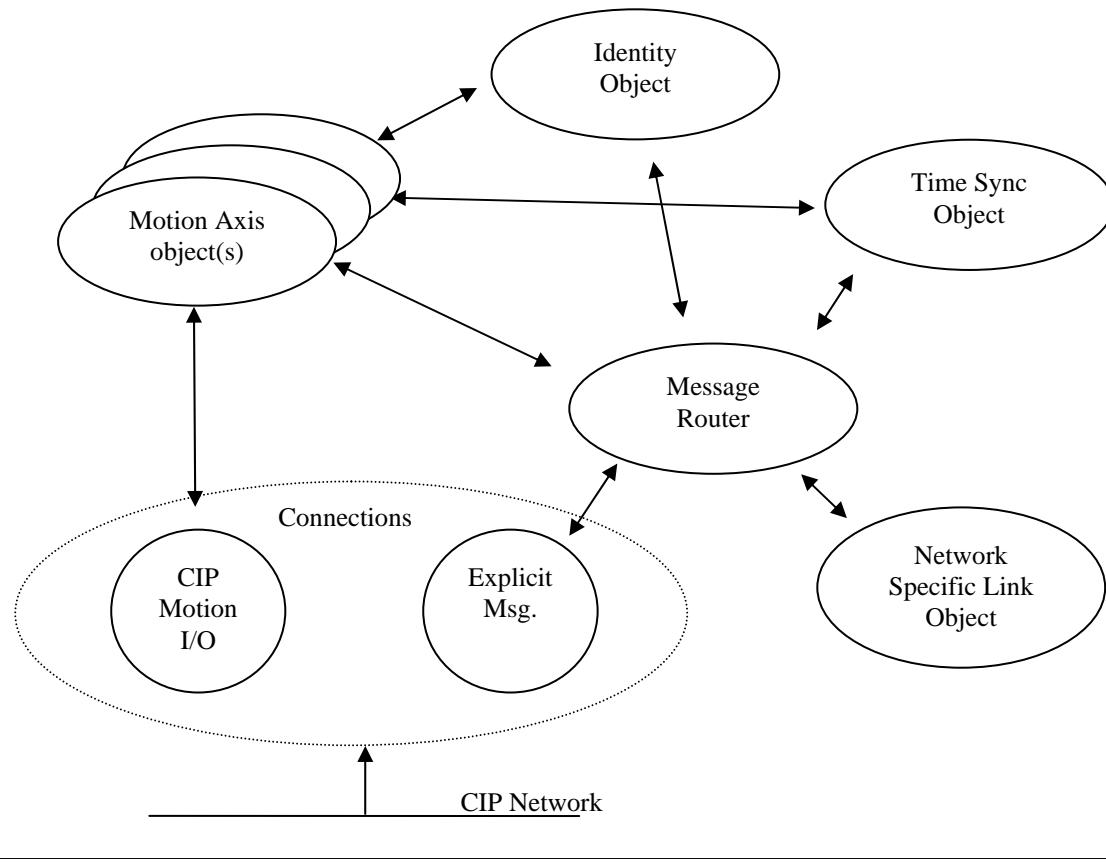
The object model shown below describes the main functional components of the CIP Motion Drive device profile defined as part of the motion control extensions to CIP. This profile is required to support motion control when connected to a motion controller that supports these motion control extensions to CIP, through a CIP network. Applicable CIP Networks would be DeviceNet, ControlNet, and EtherNet/IP. EtherNet/IP is the network of choice for high performance, synchronized multi-axis control. ControlNet and DeviceNet is only be targeted for lower performance non-synchronized drive applications such as simple variable frequency drives, velocity loop drives, and indexing drives.

The object model below also illustrates the use of multiple instances of a Motion Axis Object in multi-axis motion implementations.. Each Motion Axis Object instance governs the behavior of the associated motion axis.

A single bi-directional I/O connection to the Motion Axis Object instance provides a cyclic data path between the controller and each individual Motion Axis Object instance. This connection passes on a special data structure whose self-defining format can be used to transfer cyclic, event, and service related data.

**CIP Motion Drive, Type: 25<sub>hex</sub>**

An optional Time Sync Object is included in the object model to facilitate accurate time synchronization between motion controllers and motion axes for high performance motion control. For lower performance motion control such as V/Hz or VFD drives, the Time Sync Object is not necessary for motion axis operation.

**Figure 6-41.1 Object Model For a CIP Motion Drive****6-41.3 How Objects Affect Behavior**

The objects for this device affect the device's behavior as shown in the table below.

- The object classes present in this device
- Whether or not the class is required
- The number of instances present in each class

**Table 6-41.2 Object Effect on Behavior**

Object Class	Effect on Behavior
CIP Required Objects	See Section 6.2.2 for details.
Motion Axis object	Provides dynamic control interface to drive, motor, and feedback components that comprise an axis.
Time Sync Object	Provide absolute time synchronization services between the motion controller device and the motion axis devices on the control network.

#### 6-41.4 Defining Object Interfaces

Objects supported for the motion axis device have the interfaces listed in the table below.

**Table 6-41.3 Object Interfaces**

Object	Interface
CIP Required Objects	See Chapter 6-2.3
Motion Axis object	Message Router or Motion Axis object Class
Time Sync Object	Message Router or Motion Axis object Class

#### 6-41.5 I/O Connection Messages

This profile supports a point-to-point bi-directional I/O connection between the controller and the Motion Axis Object class. (This I/O connection is specifically referred to as the Motion I/O Connection.) The Motion Axis Object distributes the data in this connection to each instantiated Motion Axis Object instance. Therefore, the produced and consumed connection paths explicitly target Motion Axis Object class attributes (i.e. instance 0). The Consumed Connection path corresponds to the Controller-to-Device Connection while the Produced Connection path corresponds to the Device-to-Controller Connection as defined in detail the following section. The I/O Connection paths for this device are as follows:

Consumed Connection Path = Motion Axis Object class / Instance 0 / (Controller Consumed Connection Data attribute)

Produced Connection Path = Motion Axis Object class / Instance 0 / (Controller Produced Connection Data attribute)

#### 6-41.5.1 Motion Drive Connection

The following section is a description of the Motion Drive Connection format that includes the Controller-to-Device Connection and the Device-to-Controller Connection for bi-directional data transfer between a motion controller and a motion axis.

**Table 6-41.4 Controller-to-Device Connection Format**

← 32-bit Word →			
Controller-to-Device Connection Format			
Connection Header			
Instance Data Blocks			
Connection Header			
Connection Format	Format Revision	Update ID	Node Control
Instance Count	-	-	Time Data Set
Controller Update Period			
Controller Time Offset			
Controller Time Stamp			

**CIP Motion Drive, Type: 25<sub>hex</sub>**

<b>Instance Data Block</b>						
Instance Data Header						
Cyclic Data Block						
Cyclic Write Data Block						
Event Data Block						
Service Data Block						
<b>Instance Data Header</b>						
Instance Number	-	Instance Block Size	Cyclic Block Size			
Cyclic Command Block Size	Cyclic Write Block Size	Event Block Size	Service Block Size			
<b>Cyclic Data Block</b>						
Control Mode	Feedback Configuration	Axis Control	-			
Command Data Set	Actual Data Set	Status Data Set	Interpolation Control			
Cyclic Data						
<b>Cyclic Write Data Block</b>						
Cyclic Write Block ID	Cyclic Read Block ID					
Cyclic Write Data						
<b>Event Data Block</b>						
Event Checking Control						
Event Ack. ID 1	Event Ack. Status 1	Event Ack. ID 2	Event Ack. Status 2			
...						
<b>Service Data Block</b>						
Transaction ID	Service Code	-	-			
Service Specific Request Data						

**Table 6-41.5 Device-to-Controller Connection Format**

<b>Device-to-Controller Connection Format</b>			
Connection Header			
Instance Data Blocks			
<b>Connection Header</b>			
Connection Format	Format Revision	Update ID	Node Status
Instance Count	Node Alarms	Node Faults	Time Data Set
Device Update Period			
Device Time Offset			
Device Time Stamp			
Timing Diagnostic Data			

Instance Data Block			
Instance Data Header			
Cyclic Data Block			
Cyclic Read Data Block			
Event Data Block			
Service Data Block			
Instance Data Header			
Instance Num	-	Instance Block Size	Cyclic Block Size
Cyclic Actual Block Size	Cyclic Read Block Size	Event Block Size	Service Block Size
Cyclic Data Block			
Control Mode	Feedback Configuration	Axis Response	Response Status
-	Actual Data Set	Status Data Set	Axis State
Cyclic Data			
Cyclic Read Data Block			
Cyclic Read Block ID			-
Cyclic Read Data			
Event Data Block			
Event Checking Status			
Event ID	Event Status	Event Type	-
Event Position			
Event Time Stamp			
...			
Service Data Block			
Transaction ID	Service Code	General Status	Extended Status
Service Specific Response Data			

#### 6-41.5.1.1 Motion I/O Connection Overview

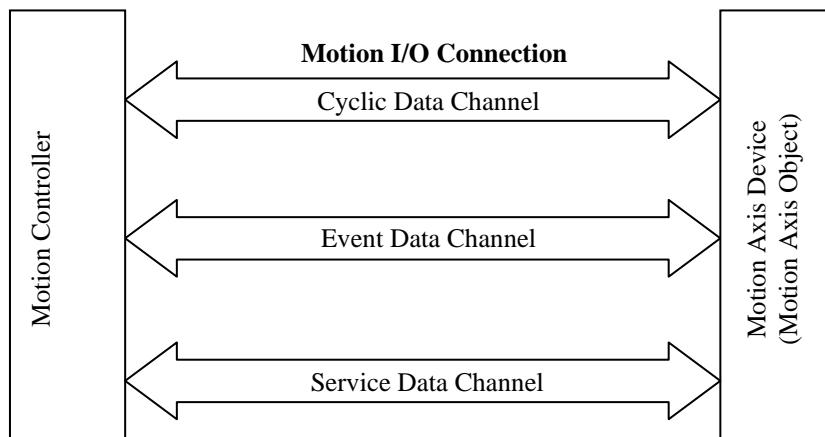
The Motion I/O Connection actually consists of two unidirectional unicast connections; one passing data from the motion controller to the motion axis and the other passing data from the motion axis to the motion controller.

#### 6-41.5.1.2 Motion Connection Structure

Both the Motion I/O Connection data structures (controller-to-device and device-to-controller) begin with a Connection Header that includes a 32-bit time stamp, followed by a series of data blocks for each axis instance supported by the device.

**CIP Motion Drive, Type: 25<sub>hex</sub>**

Each instance data block within the Motion I/O Connection packet consists of three sets of data blocks associated with the cyclic, event, and service data channels. The size of these data blocks for a given update is variable and determined by the connection and instance data block headers. From the motion axis device's perspective, these three distinct data channels have different data processing priorities as illustrated in the diagram below.

**Figure 6-41.2 Motion I/O Connection Channels**

The specific functionality of these three data channels is as follows:

- **Cyclic Data Channel** – carries cyclic data blocks that are sampled or calculated every Controller Update Period and synchronized with other nodes in the motion control system through use of distributed System Time. Cyclic data is high priority data that must be immediately processed and applied to the motion axis within one DeviceUpdate Period.
- **Event Data Channel** - carries event data associated with motion axis event(s) (e.g. registration, homing, etc.) that have occurred within the last Controller Update Period. Event data is medium priority and must be processed and applied within one Controller Update Period.
- **Service Data Channel** – carries data associated with service requests to read or write attribute values of the Motion Axis Object as part of on-line configuration and diagnostic functionality, as well as service requests to affect Motion Axis Object behavior as part of controller instruction execution. Service data has lowest priority and is typically buffered and processed as a background task. There is no guarantee that a service request will be processed within Controller Update Period.

Taken together, these three data channels provide a comprehensive motion controller to motion axis device data connection solution for industrial motion control.

#### **6-41.5.2 Controller-to-Device Connection**

To facilitate a detailed description of each of its constituent data elements, the Controller-to-Device Connection is organized as follows:

**Table 6-41.6 Controller-to-Device Connection Format**

Controller-to-Device Connection Format
Connection Header
Instance Data Blocks

### 6-41.5.2.1 Controller-to-Device Connection Header

The Controller-to-Device Connection Header contains critical axis configuration information needed to parse the Instance Data Blocks. The fixed portion of the connection header is defined as follows.

**Table 6-41.7 Connection Header**

Connection Header			
Connection Format	Format Revision	Update ID	Node Control

- **Connection Format:** This enumerated byte determines the format of the connection according to the following definition:

**Table 6-41.8 Connection Format**

Connection Format	
Bit 7	Bit 0
(Reserved)	Connection Type

- **Connection Type:** This 4-bit enumeration defines the connection type as shown below. Valid values for a Controller-to-Device Connection are 2 and 6. Fixed connections have a fixed connection size during operation to support other CIP networks like ControlNet and DeviceNet, and typically associated with a simple single axis device that does not support time synchronization services. Variable connections allow the connection data structure to vary in size during operation and is targeted for high-performance motion and CIP networks like EtherNet/IP.

0 = Reserved  
 1 = Reserved  
 2 = Fixed Controller-to-Device Connection  
 3 = Fixed Device-to-Controller Connection  
 4 = Reserved  
 5 = Reserved  
 6 = Variable Controller-to-Device Connection  
 7 = Variable Device-to-Controller Connection  
 8-15 = Reserved

- **Format Revision:** This release of the specification defines Format Revision 1. Devices utilizing this version of the specification will only recognize Format Revision 1. This value is incremented by 1 for every revision of the Controller-to-Device Connection format that impacts the interface. The Format Revision allows newer motion axis devices to support the connection formats generated by an older controller. It also allows older motion axis devices to recognize a newer connection format from a controller that it cannot support and generate an appropriate error.

**CIP Motion Drive, Type: 25<sub>hex</sub>**

- Update ID:** This cyclic transaction number is incremented every update period. The Update ID is like the CIP sequence count and is used by the motion axis device to determine whether the connection buffer contains fresh data. If the Update ID has not changed, the motion axis device attempts to ride through the missed update by extrapolation based on previous trajectory until fresh data arrives. In the case where the motion axis is not synchronized, or does not support time synchronization services, the time stamp data is either not included or invalid, so the Update ID is the only way for the motion axis device to detect new connection data. In this case, the Controller-to-Device connection Update ID is applied to the next Device-to-Controller connection Update ID.
- Node Control:** This element is applied to the Motion Axis Object class attribute, Node Control, which is used to control the state of the associated drive communications node. See Motion Axis Object for detail.

**6-41.5.2.1.1 Fixed Connection Header**

If the Connection Format is a Fixed Controller-to-Device Connection the above header is immediately followed by the instance data block.

**6-41.5.2.1.2 Variable Connection Header**

If the Connection Format is a Variable Controller-to-Device Connection then the connection header contains additional fields related to multi-axis motion device addressing and time stamping.

**Table 6-41.9 Connection Header**

Connection Header			
Connection Format	Format Revision	Update ID	Node Control
Instance Count	-	-	Time Data Set
Controller Update Period			
Controller Time Offset			
Controller Time Stamp			

- Instance Count:** This value reflects the number of instance data blocks present in the Controller-to-Device Connection data structure.
- Time Data Set:** This bit-mapped byte contains flags that determine the usage and format of the controller and motion axis device timing information. The Time Data Set value sent by the controller in the Controller-to-Device connection becomes the Time Data Set value sent by the motion axis device in the Device-to-Controller connection. The value of this element is transferred to the Motion Axis Object attribute, Command Data Set.

**Table 6-41.10 Time Data Set**

Bit	Definition	Syntax
0	Update Period	0 = no update period 1 = update period included
1	Time Stamp	0 = no time stamp & offset 1 = time stamp & offset included
2	Time Diagnostic Data	0 = no diagnostic data 1 = diagnostic data included
3-7	Reserved	

**CIP Motion Drive, Type: 25<sub>hex</sub>**

- **Controller Update Period:** This 32-bit element specifies the update period of the Controller-to-Device connection in nanoseconds. The value of this element is transferred to the Motion Axis Object class attribute of the same name. When the connection is synchronized, the Controller Update Period can be used by the motion axis device along with the Controller Time Stamp to check for missed or late updates. If the difference between the last Controller Time Stamp and the current local motion axis device time stamp exceeds the maximum value given by,

Max Delay = Controller Update Delay High Limit \* Controller Update Period,

the Controller Update Fault bit is set in the Node Faults class attribute of the Motion Axis Object. An optional Controller Update Delay Low Limit attribute is also available. If the difference between the last Controller Time Stamp and the current local motion axis device time stamp exceeds the maximum value given by,

Max Delay = Controller Update Delay Low Limit \* Controller Update Period,

the Controller Update Alarm bit is set in the Node Alarm class attribute of the Motion Axis Object.

When the connection is synchronized and fine interpolation is enabled, the Controller Update Period is also used by the drive along with the Controller Time Stamp value to compute new coefficients to the interpolation/extrapolation polynomial.

In the case where the connection is not synchronized, no late update checks are performed by the motion axis device. In this case the command data should be applied immediately, which means applying the data immediately as a command reference.

- **Controller Time Offset:** This element represents the 64-bit System Time Offset value associated with the Controller Time Stamp that follows. The value of this element is transferred to the Motion Axis Object class attribute of the same name. The Controller Time Offset value is used by the motion axis device to determine if System Time as defined in the motion controller is skewed relative to System Time in the motion axis device. Normally, System Time between the motion axis device and the motion controller are closely matched at any given time. But every few seconds, according to the IEEE-1588 protocol, the time master of the system can correct the System Time reference by a significant amount of time, perhaps an hour. Indeed, even the master itself can change as a higher quality time master is discovered by the system. These step changes to the System Time reference propagate through to the various devices on the network in such a way that the controller and the motion axis device are running with skewed System Time values. The Controller Time Offset value can be used together with the System Time Offset for the motion axis device to both determine if System Time between the motion axis device and the motion controller is skewed and if so to compensate for the skew.

- **Controller Time Stamp:** This element represents the 64-bit System Time value at the beginning of the Controller Update Period when the motion controller's update timer event occurred. It is calculated by the motion controller as the sum of its local clock value when update timer event occurred and the motion controller's System Time Offset value. The Controller Time Stamp is therefore directly associated with the command data contained in the connection. The value of this element is transferred to the Motion Axis Object class attribute of the same name. Taken together with the Controller Update Period and the Command Target Time (discussed later), the motion axis device has all the information it needs compute command interpolation/extrapolation polynomials to correct command data values for differences between the motion axis device and controller update timing. These differences can occur when the Controller Update Period is not an integer multiple of the Device Update Period or when the motion axis device updates are phase shifted relative to the motion controller.

#### 6-41.5.2.2 Instance Data Blocks

After the Connection Header are one or more Instance Data Blocks as determined by the above Instance Count. The Instance Data Block has the following basic structure:

**Table 6-41.11 Instance Data Block**

Instance Data Block
Instance Data Header
Cyclic Data Block
Event Data Block
Service Data Block

##### 6-41.5.2.2.1 Instance Data Header

The Instance Data Header contains critical axis configuration information needed to parse and apply the data contained in the three data channels. This header is only included in the Variable Connection format to accommodate multi-axis applications. Information within the header can be used by the motion axis device's communications interface to copy the individual data blocks into separate fixed memory locations for processing.

If configured for a Fixed Connection format, only the Cyclic Data Block for a single axis instance is supported so there is no need for any information to specify instance number or block sizing. The Instance Data Header is therefore not included.

**Table 6-41.12 Instance Data Header**

Instance Data Header			
Instance Number	Reserved	Instance Block Size	Cyclic Block Size
Cyclic Actual Block Size	Cyclic Write Block Size	Event Block Size	Service Block Size

- **Instance Number:** This is the number that identifies the specific Motion Axis Object instance the following instance data block applies to. Motion Axis Object instances are created as a contiguous series of instance numbers starting with instance 1. Within the connection data structure the Instance Numbers for each consecutive instance data block must be an ordinal sequence, i.e. 1, 2, 3, etc... (Note, instance 0 is defined as the class instance and not generally used as part of the connection data.) Thus, in theory, up to 255 instances can be serviced by the Motion I/O Connection.

**CIP Motion Drive, Type: 25<sub>hex</sub>**

- **Instance Block Size:** This value represents the size of the Instance Data Block in 32-bit words including the header. The Instance Block Size is useful when the motion axis device wants to directly access the next instance data block without having to add the sizes of the cyclic, event, and service blocks.
- **Cyclic Block Size:** This value represents the size to the Cyclic Data Block in 32-bit word units including the header.
- **Cyclic Command Block Size:** This value represents the size to the Cyclic Command Data Block in 32-bit word units including the header.
- **Cyclic Write Block Size:** This value represents the size to the Cyclic Write Data Block in 32-bit word units including the header. If the Cyclic Write Block Size of 0 indicates, the motion axis device does not need to support the Cyclic Read/Write functionality.
- **Event Block Size:** This value represents the size to the Event Data Block in 32-bit word units including the header. If the Event Block Size is 0, the motion axis device does not need to support the event functionality.
- **Service Block Size:** This value represents the size to the Service Data Block in 32-bit word units including the header. A Service Block Size value of 0 indicates there is no service request to process.

**6-41.5.2.2.2 Cyclic Data Block**

The Cyclic Data Header at the top of the Cyclic Data Block is always included regardless of the connection format. This header contains key elements related to the content of the Cyclic Data Block of both the Controller-to-Device Connection and Device-to-Controller Connection, and, the context of the data as determined by the Control Mode and Feedback Configuration. The header also provides a mechanism to control the state of the targeted motion axis.

**Table 6-41.13 Cyclic Data Header**

Cyclic Data Header			
Control Mode	Feedback Configuration	Axis Control	-
Command Data Set	Actual Data Set	Status Data Set	Interpolation Control
Cyclic Data			

- **Control Mode:** An 8-bit enumerated value that determines the control mode context of the command data as presently configured in the motion controller. This value can be changed while on-line and even while the motion axis is in the Running state. If a particular Control Mode transition is not supported by the motion axis device, an Illegal Mode Change exception is generated that can be configured to perform any one of a number of actions in response to the illegal transition.

The value of this element is transferred to the Motion Axis Object attribute of the same name. The complete Control Mode attribute definition can be found in the attribute tables in the Motion Axis Object definition in Chapter 5.

- **Feedback Configuration:** This 8-bit enumerated value that indicates the feedback context of the command data as presently configured in the motion controller. Command data can be referenced to feedback counts of either Feedback 1 or Feedback 2, or motor units for sensor-less operation. This value can be changed while on-line and even while the motion axis is in the Running state. If a particular Feedback Configuration transition is not supported by the motion axis device, an Illegal Mode Change exception is generated that can be configured to perform any one of a number of actions in response to the illegal transition.

**CIP Motion Drive, Type: 25<sub>hex</sub>**

The value of this element is transferred to the Motion Axis Object attribute of the same name. The complete Feedback Configuration attribute definition can be found in the attribute tables in the Motion Axis Object definition in Chapter 5.

- **Axis Control:** This 8-bit enumerated code that can be used to directly initiate motion axis state change operations that do not require either passing or returning motion axis parameters, and therefore, do not require a CIP service to initiate. Valid enumerations for this data element are shown below:

**Table 6-41.14 Axis Control**

Request Code	Requested Operation
0	No Request
1	Enable Request
2	Disable Request
3	Shutdown Request
4	Shutdown Reset Request
5	Abort Request
6	Fault Reset Request
7-127	Reserved
128-255	Vendor Specific

- **Command Data Set:** This bit mapped value has a bit defined for each possible real-time command reference. Command data appears in the same order in the Command Data Set as the bit numbers, so Command Position would appear before Command Torque in the real-time data structure of the Controller-to-Device Connection. The value of this element is transferred to the Motion Axis Object attribute, Command Data Set.

**Table 6-41.15 Command Data Set**

Bit	Controller Command Data Element	Data Type
0	Command Position	DINT
1	Command Velocity	REAL
2	Command Acceleration	REAL
3	Command Torque	REAL
4	Position Trim	DINT
5	Velocity Trim	REAL
6	Acceleration Trim	REAL
7	Torque Trim	REAL

The above Controller Command Data Elements apply to the Controller-to-Device Connection's cyclic data structure and map to corresponding attributes in the Motion Axis Object as shown in the table below.

The units of the Command Data Elements match the units defined for the associated Motion Axis Object attribute.

**CIP Motion Drive, Type: 25<sub>hex</sub>****Table 6-41.16 Command Data Element to Motion Axis Object Attribute Mapping**

<b>Bit</b>	<b>Command Data Element</b>	<b>Motion Axis Object Attribute</b>
0	Command Position	Controller Position Command
1	Command Velocity	Controller Velocity Command
2	Command Acceleration	Controller Acceleration Command
3	Command Torque	Controller Torque Command
4	Position Trim	Position Trim
5	Velocity Trim	Velocity Trim
6	Acceleration Trim	Acceleration Trim
7	Torque Trim	Torque Trim

It is the job of the motion controller to insure that the necessary Command Data Elements are included in the connection data to support the specified Control Mode.

- **Actual Data Set:** This bit-mapped value has a bit defined for each possible real-time actual data attribute that is to be included in the Actual Data Set of the Device-to-Controller connection's Instance Data Block in the next update. Actual data appears in the same order as the bit numbers, so Actual Position would appear before Actual Torque in the Actual Data Set structure. Using this mechanism, the contents of the Actual Data Set may be changed at any time during motion axis device operation. The value of this element is transferred to the Motion Axis Object attribute, Feedback Data Set.

**Table 6-41.17 Actual Data Set**

<b>Bit</b>	<b>Actual Data Element Produced</b>	<b>Data Type</b>
0	Actual Position	DINT
1	Actual Velocity	REAL
2	Actual Acceleration	REAL
3	Actual Torque	REAL
4	Actual Current	REAL
5	Actual Voltage	REAL
6	Actual Frequency	REAL
7	Reserved	

The above Actual Data Elements map to corresponding attributes in the Motion Axis Object as shown in the table below. The units of the Actual Data Elements match the units defined for the associated Motion Axis Object attribute.

**Table 6-41.18 Actual Data Element to Motion Axis Object Attribute Mapping**

<b>Bit</b>	<b>Actual Data Element</b>	<b>Motion Axis Object Attribute</b>
0	Actual Position	Position Feedback
1	Actual Velocity	Velocity Feedback
2	Actual Acceleration	Acceleration Feedback
3	Actual Torque	Torque Reference
4	Actual Current	Iq Current Reference
5	Actual Voltage	Output Voltage
6	Actual Frequency	Output Frequency
7	Reserved	

**CIP Motion Drive, Type: 25<sub>hex</sub>**

- Status Data Set:** This bit-mapped byte contains flags that determine the contents of the Status Data Set of the Device-to-Controller Connection's Instance Data Block in the next update. Status data appears in the same order as the bit numbers, so Drive Fault Status would appear before, say, Drive Alarm Status data in the Status Data Set structure. Multiple attributes comprising a selected Status Data Element are transmitted in the order listed from top to bottom, so Drive Fault Code is transmitted before Mfg Drive Fault Code. The definitions of each of these Status Data Elements can be found by looking up the corresponding Motion Axis Object attribute in the Motion Axis Object specification. The value of this element is transferred to the Motion Axis Object attribute, Status Data Set.

**Table 6-41.19 Status Data Set Configuration**

Bit	Status Data Element Produced	Data Type
0	Initialization Fault Code	USINT
	Initialization Fault Code - Mfg	USINT
	Axis Fault Code	USINT
	Axis Fault Code - Mfg	USINT
	Axis Alarm Code	USINT
	Axis Alarm Code - Mfg	USINT
	Start Inhibit Code	USINT
	Start Inhibit Code - Mfg	USINT
	Axis Fault Time Stamp	LINT
1	Axis Fault Status	LWORD
	Axis Fault Status - Mfg	LWORD
2	Axis Alarm Status	LWORD
	Mfg Drive Alarm Status	LWORD
3	Start Inhibit Status	WORD
	Start Inhibit Status - Mfg	WORD
4	Axis Status	DWORD
	Mfg Drive Status	DWORD
	Axis I/O Status	DWORD
	Mfg Drive I/O Status	DWORD
5	Reserved	Reserved
6	Reserved	Reserved
7	Vendor Specific	Vendor Specific

- Interpolation Control:** The byte contains information needed to control the fine interpolation algorithm and determine the target time of the command data to the Axis Control structure. The value of this element is transferred to the Motion Axis Object attribute of the same name.

**Table 6-41.20 Interpolation Control**

Bit	Definition	Syntax
0-1	Command Target Time	0 = Immediate 1 = Extrapolate (+1 Update Period) 2 = Interpolate (+2 Update Periods)
3-5	Reserved	
6-7	Vendor specific	

**CIP Motion Drive, Type: 25<sub>hex</sub>**

- **Command Target Time** - This 2-bit integer defines a specific time relative to the Connection Time Stamp that the Command Data is targeted for, as defined by the controller's motion planner. The absolute command target time is the sum of the Controller Time Stamp from the motion controller and the product, Command Target Time \* Controller Update Period.

A Command Target Time of 0 implies that the Command Data is targeted for the beginning of the current update cycle and, thus, needs to be applied to the control structure immediately. In this case there is no need for any fine interpolation. This situation can occur when the Controller Update Period is significantly shorter than the Device Update Period, or when the controlled motion of the axis is nearly constant during the span of the Controller Update Period.

A Command Target Time of 1 implies the target for the Command Data is the next Connection Update timer event. In this case, the command interpolator functions primarily as an extrapolator that estimates the next Command Data value based on the present trajectory of the axis. This is a typical setting when the Controller Update Period is comparable to the Device Update Period or when the controlled motion of the axis is relatively constant during the span of the Controller Update Period.

A Command Target Time of 2 implies the target for the Command Data is two Connection Update timer events from the Connection Time Stamp. In this case, the command interpolator can compute a smooth trajectory based on the current dynamics of the motor to reach the Command Data value at the targeted time. This is true fine interpolation and is applicable when the Controller Update Period is significantly larger than the Device Update Period.

- **Cyclic Command Data:** The Cyclic Command Data contains high priority data that needs to be applied to the associated motion axis during the next notion axis update. This block consists of command data elements that are applied as references to the motion axis' control algorithms and explicitly determined by the Command Data Set element in the Cyclic Command Data Header.

#### **6-41.5.2.2.3 Cyclic Write Data Block**

The Cyclic Write Data Block can be used to synchronously update one or more targeted Motion Axis Object configuration parameters within the device. This mechanism can be used in conjunction with a Function Block program to implement sophisticated outer loop control, gain scheduling, and dynamic limiting algorithms. Unlike service channel Set Axis Attribute service requests, which may take several axis update cycles to process, the Cyclic Write Data mechanism guarantees the targeted parameter is applied at the next available axis update.

The Cyclic Write Data Block is only supported in the Variable Connection format.

**Table 6-41.21 Cyclic Write Data Block**

<b>Cyclic Write Data Block</b>	
Cyclic Write Block ID	Cyclic Read Block ID
Cyclic Write Data	

The associated header for this block contains key elements related to the content of the Cyclic Write Data Block.

- **Cyclic Write Block ID:** This 16-bit ID determines the pre-defined Cyclic Write Block structure to apply to the Cyclic Write Data for this update. Cyclic Write Block structures are defined using the Set Cyclic Write Data List service. A successful response to this service includes a new Cyclic Write Block ID that can be used in the next connection update to pass cyclic data in this format. The previous Cyclic Write Block ID and its associated structure must be maintained until the controller requests transmission of the new Cyclic Write Block ID at which point the old Cyclic Write Block ID and structure definition are no longer required.
- **Cyclic Read Block ID:** This 16-bit ID determines the pre-defined Cyclic Read Block structure to apply to the Cyclic Read Data for next Device-to-Controller connection update. Cyclic Read Block structures are defined using the Set Cyclic Read Data Block service. A successful response to this service includes a new Cyclic Read Block ID that can be used in the next connection update to allow the device to use the new Cyclic Read Data format for next available Device-to-Controller connection update.
- **Cyclic Write Data:** The Cyclic Write Data contains high priority data that needs to be applied to the associated motion axis instance during the next axis update. This block consists of parametric data elements that are applied to Motion Axis Object attributes that are used by the control algorithms. The contents of the Cyclic Write Data are explicitly determined by the structure identified by the Cyclic Write Block ID found in the Cyclic Write Data Header.

#### 6-41.5.2.3 Event Data Block

The Event Data Block is used to convey information regarding the event channel. In particular the Event Data Block for the Controller-to-Device Connection is used to control the arming of event checking functions in the device as well as acknowledge receipt of event notifications from the device that are sent via the Device-to-Controller Connection's Event Data Block.

The Event Data Block for the Controller-to-Device Connection has the following format.

**Table 6-41.22 Event Data Block**

Event Data Block			
Event Checking Control			
Event Ack. ID 1	Event Ack. Status 1	Event Ack. ID 2	Event Ack. Status 2
...			

- **Event Checking Control:** This 32-bit word is copied into the Motion Axis Object attribute of the same name that is used to enable various drive inputs, e.g. marker and registration inputs, to generate events. When these events occur, the device captures both the time and exact axis position when the event occurred. The last 4 bits of the Event Checking Control is a binary value that specifies the number of active events, which is literally the number of Event Acknowledge IDs listed in this Event Data Block. A complete definition for the Event Checking Control attribute can be found in the Motion Axis Object definition in Chapter 5.

The Event Control mechanism works as follows:

1. Controller sets the appropriate Event Checking bit to look for a specific event condition to occur. Multiple bits in the Event Checking Control word may be set at any given time.

**CIP Motion Drive, Type: 25<sub>hex</sub>**

2. Drive detects the Event Checking Control request bit is set and initiates the requested event checking action.
3. Drive sets the corresponding Event Checking Status bit to acknowledge that the device is now actively checking for the specified event condition.
4. Controller sees the Event Checking Status bit from the device is set indicating that specified event trigger is “armed” at the device. Starting with this update the controller can process any event notifications from the device that match the specified event condition.
5. Motion axis device detects the specified event condition, increments the Event ID, and sends an Event Notification Data Block with this ID to the controller. If the associated Auto-rearm bit is clear in the Event Checking Control word, the device discontinues checking for the specified event. However, the Event Checking Status bit remains set until the Event Checking Request bit is cleared by the motion controller. Note that notifications for other events can be sent in the same update.
6. Motion controller processes the Event Notification, sends the Event Acknowledge to the device and, if Auto-rearm is not enabled, clears the associated Event Checking bit for the next Controller-to-Device connection update.
7. Motion axis device detects the Event Checking Request bit is clear and clears the corresponding Event Checking Status bit prior to the next Device-to-Controller connection update.
8. Controller is able to set the Event Checking bit again as soon as it sees the associated Event Checking Status bit has been reset. So, the minimum time between when the Event Notification was received by the controller to the time when the device is rearmed for the next event is one Controller Update Period.

In the case of a Registration event where Auto-rearm Event Checking is requested, the event handling sequence is as follows:

1. Motion controller sets the appropriate Event Checking bit to look for a specific event condition to occur and also sets the Auto-rearm bit for that event condition.
2. Motion axis device detects the Event Checking Control request bit is set and initiates the requested event checking action.
3. Motion axis device sets the corresponding Event Checking Status and Auto-rearm bits to acknowledge that the device is now actively checking for the specified event condition.
4. Motion controller sees the Event Checking Status bit from the motion axis device is set indicating that specified event trigger is “armed” at the device. Starting with this update the motion controller can process any event notifications from the device that match the specified event condition.

**CIP Motion Drive, Type: 25<sub>hex</sub>**

5. Motion axis device detects the specified event condition, increments the Event ID, and sends an Event Notification Data Block with this ID to the motion controller. Since the corresponding Auto-rearm bit is set, the device continues checking for the specified event. The Event Checking Status bit remains set until the Event Checking Request bit is cleared by the motion controller. If the device detects another specified event condition prior to transmission of the first event to the controller, it increments the Event ID again, and append another Event Notification with this ID to the Event Data Block to the motion controller. Since the corresponding Auto-rearm bit is set, the device continues checking for the specified event.
  6. Motion controller processes the Event Notification, sends the Event Acknowledge to the device but, since the Auto-rearm bit is enabled, it leaves the associated Event Checking bit set for the next Controller-to-Device connection update.
  7. Motion axis device receives the Event Acknowledge from the motion controller. It can now send another Event Notification if the specified event has occurred again. Otherwise...
  8. Motion axis device continues checking for the specified event condition and sends an Event Notification Data Block to the controller whenever the event condition occurs. The Event Checking Status bit remains set until the Event Checking Request bit is cleared by the motion controller. With the Auto-rearm feature, event checking is continuously enabled, insuring that no registration events are missed during the normal one cycle delay in re-arming the event checking mechanism. The down side of Auto-rearm feature is that it can generate a multitude of uninteresting events to process, in fact, multiple specified events per Controller Update Period. Therefore, these events must either be buffered in the motion controller in an event array attribute or filtered based on the event data. The later case is how Windowed Registration functionality is implemented; the Windowed Registration feature checks each registration event that arrives from the motion axis device to see if the Event Position yields an absolute position value that is within the configured Registration Window. If the computed registration position is outside the window, the event is thrown away. If the computed registration position is within the window the registration position and registration time stamp is stored in the motion controller.
- **Event Acknowledge ID:** Transaction number assigned to this event by the original event notification. Each event is assigned a new Event ID by incrementing the current Event ID stored in the motion axis device. Using the Event ID, the device is able to match the event acknowledgement to the appropriate event notification to complete the event data transaction.
  - **Event Acknowledge Status:** Enumerated value indicating motion controller response to the event. A value of 0 indicates that the event was successfully processed. A non-zero value indicates that an error occurred in the event processing and the event must be resent.

**6-41.5.2.4 Service Data Block**

The service data block allows one service request per instance to be sent to the motion axis device in a given update. The service request requires a specific service response from the device indicating success or an error. In some cases the response service contains requested data. In any case, the service request data persists in the Controller-to-Device Connection data structure until the controller receives the associated service response from the device.

**CIP Motion Drive, Type: 25<sub>hex</sub>**

Each service request is represented by a block of data organized as shown below.

Design Note: By design, of the first 4 bytes of the service data block do not follow the traditional CIP standard messaging format. That is primarily because this connection structure is, fundamentally, a CIP Implicit I/O connection, not an Explicit Messaging connection. However, in the case of a Fixed Connection format, the Service Specific Request Data defined below is sent via an Explicit Messaging connection and follows the CIP rules for explicit service request format

**Table 6-41.23 Service Data Block**

Service Data Block			
Transaction ID	Service Code	-	-
Service Specific Request Data			

- **Transaction ID:** Transaction number assigned to this service request by the motion controller. Each service request is assigned a new Transaction ID by incrementing the current Transaction ID stored in the controller. Using the Transaction ID, the controller is able to match the service response to the appropriate service request and complete the service transaction.
- **Service Code:** Identifier that determines the object specific service request that follows. The list of supported Service Codes can be found in the Object Specific Services section of this document. CIP Common services are not applicable to the Service Data Block.
- **Service Specific Request Data:** The format and syntax of the Service Specific Request Data depends on the specified Service Code. This is true regardless of whether the service specific request data is passed in the Controller-to-Device connection or as part of an Explicit messaging connection.

**6-41.5.3 Device-to-Controller Connection**

Like the Controller-to-Device Connection data structure described above, the Device-to-Controller Connection is organized as follows:

**Table 6-41.24 Motion Device-to-Controller Connection Format**

Motion Device-to-Controller Connection Format	
Connection Header	
Instance Data Blocks	

**6-41.5.3.1 Device-to-Controller Connection Header**

The Device-to-Controller Connection Header contains critical axis configuration information needed to parse the Device-to-Controller connection data block. The fixed portion of the connection header is defined as follows:

**Table 6-41.25 Connection Header**

Connection Header			
Connection Format	Format Revision	Update ID	Node Status

**CIP Motion Drive, Type: 25<sub>hex</sub>**

- **Connection Format:** Same as Controller-to-Device definition except the required value for the Connection Type is either 3, indicating a Fixed Device-to-Controller connection type or 7, indicating a Variable Device-to-Controller connection type.

0 = Reserved  
 1 = Reserved  
 2 = Fixed Controller-to-Device Connection  
 3 = Fixed Device-to-Controller Connection  
 4 = Reserved  
 5 = Reserved  
 6 = Variable Controller-to-Device Connection  
 7 = Variable Device-to-Controller Connection  
 8-15 = Reserved

- **Format Revision:** The Format Revision number is 1 for any device that utilizes this version of the specification. This value is to be incremented by 1 for every revision of the Device-to-Controller Connection format that impacts the interface. The Format Revision allows newer controllers to support the connection formats generated by older drives. It also allows older controllers to recognize a newer connection format from a drive that it cannot support and generate an appropriate error to its application.
- **Update ID:** The Device-to-Controller connection Update ID should match the Update ID of the preceding Controller-to-Device Update ID and therefore must be incremented every update period. In the case where the associated Controller-to-Device packet is lost or late, the Device-to-Controller Update ID must be incremented as if the Controller-to-Device packet had arrived on time. This allows the motion control system to ride through a lost or missed Controller-to-Device packet and maintain synchronization with matching Update IDs.

The Update ID is like the CIP message sequence count and is used by the motion controller to determine whether the connection buffer contains fresh data. If the Update ID has not changed, the controller attempts to ride through the missed update by extrapolation based on previous trajectory until a fresh update arrives. In the case where the axis is not synchronized, or does not support time synchronization services, the time stamp data is either not included or invalid, so no ride through extrapolation is attempted. In that case, the Update ID is also the only way for the controller to detect new connection data.

- **Node Status:** Contains bits used to indicate the status of the associated motion axis device. The value of this element is derived from the Motion Axis Object class attribute of the same name. For details refer to the Motion Axis Object.

**6-41.5.3.1.1 Fixed Connection Header**

If the Connection Format is a Fixed Device-to-Controller Connection the above header is immediately followed by the instance data block.

**6-41.5.3.1.2 Variable Connection Header**

If the Connection Format is a Variable Device-to-Controller Connection then the connection header contains additional fields related to multi-axis addressing and time stamping.

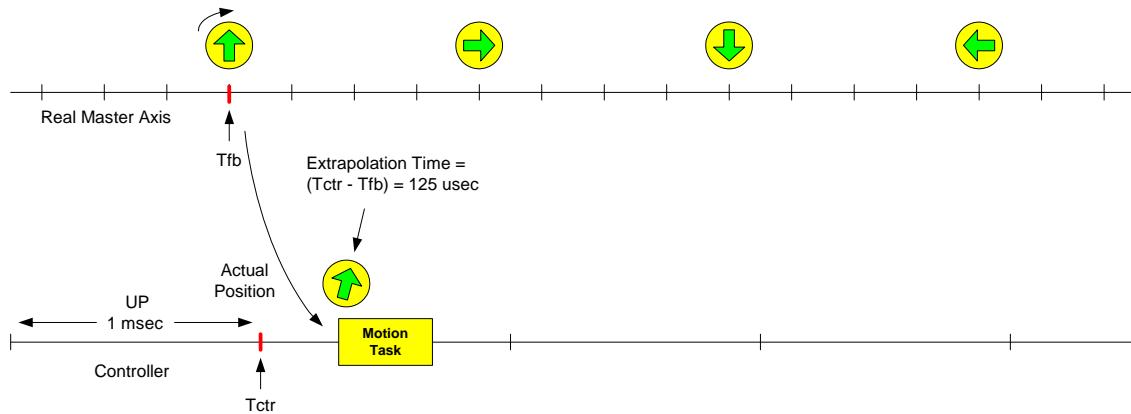
**Table 6-41.26 Connection Header**

Connection Header			
Connection Format	Format Revision	Update ID	Node Control
Instance Count	Node Alarms	Node Faults	Time Config
Device Update Period			
Device Time Offset			
Device Time Stamp			
Data Received Time Stamp			
Data Transmit Time Stamp			
Sync Variance			

- **Instance Count:** Same as Controller-to-Device definition.
- **Node Alarms:** Contains bits used to indicate the alarm conditions of the associated motion axis device. The value of this element is derived from the Motion Axis Object class attribute of the same name. For details refer to the Motion Axis Object.
- **Node Faults:** Contains bits used to indicate the fault conditions of the associated motion axis device. The value of this element is derived from the Motion Axis Object class attribute of the same name. For details refer to the Motion Axis Object.
- **Time Data Set:** Same as Controller-to-Device definition.
- **Device Update Period:** This element represents the current period between motion axis control calculations that apply the command data. This value is used by the motion controller to configure the behavior of the fine interpolation blocks that are applied to the command data. If the Controller Update Period is much longer than the Device Update Period, fine interpolation is generally applied. When the Controller Update Period is comparable to the Device Update Period, fine interpolation is unnecessary and only extrapolation is required to perform time stamp corrections.
- **Device Time Offset:** This element represents the 64-bit System Time Offset value associated with the Device Time Stamp that follows. The Device Time Offset value is used by the motion controller to determine if System Time as defined in the motion axis device is skewed relative to System Time in the controller. Normally, System Time between the motion axis device and the motion controller are closely matched at any given time. But every few seconds, according to the IEEE-1588 protocol, the time master of the system can correct the System Time reference, sometimes by a significant amount of time, perhaps as much as an hour. Indeed, even the master itself can change as a higher quality time master is discovered by the system. These step changes to the System Time reference propagate through to the various devices on the network in such a way that the motion controller and the motion axis device are running with skewed System Time values. The Device Time Offset value can be used together with the System Time Offset for the motion controller to both determine if System Time between the motion axis device and the motion controller is skewed and if so to compensate for the skew.

- **Device Time Stamp:** This time stamp value represents the 64-bit System Time value, in nanoseconds, when the motion axis' update timer event occurred that is associated with the actual data in the connection structure, e.g. when the actual data was captured. It is calculated by the motion axis device as the sum of its local clock value when update timer event occurred and the device's System Time Offset value. With the Device Time Stamp, the motion controller has all the information it needs to correct actual data values for differences between the motion axis device and motion controller update timing that result when the Controller Update Period is not an integer multiple of the Device Update Period or when the device updates are phase shifted relative to the controller. It is assumed in this timing model that the Device Time Stamp is registered to the beginning of the Device Update Period and is also the time when feedback was last captured. In the case where the Device Time Stamp does not match the local update time stamp of the motion controller, the controller extrapolates the actual response data value based on trajectory to correspond to the controller's time stamp. The timing diagram below illustrates how axis position data from the motion axis device is adjusted by the motion controller based on the relative time stamps between the motion axis device and the controller.

**Figure 6-41.3 Adjustment of Actual Position Data based on Device Time Stamp**



**Data Received Time Stamp:** This element represents the 64-bit System Time value at the moment that the last Control-to-Device connection data arrived at the motion axis device and was ready for processing. The time stamp units are nanoseconds. This value, when combined with the motion controller's corresponding data transmit time stamp, can be used by the motion controller to generate Control-to-Device connection data delivery statistics.

**Data Transmit Time Stamp:** This element represents the 64-bit System Time value at the moment that the Device-to-Controller connection data is transmitted to the motion controller. The time stamp units are nanoseconds. This value, when combined with motion controller's corresponding data received time stamp, can be used by the controller to generate Device-to-Controller connection data delivery statistics.

**Sync Variance:** This element represents the current statistical variation of the System Time Offset value from the mean System Time Offset value.

#### 6-41.5.3.2 Instance Data Blocks

After the Connection Header are one or more Instance Data Blocks as determined by the above Instance Count. The Instance Data Block is very similar to that of the Controller-to-Device Connection and has the following basic structure:

**Table 6-41.27 Instance Data Block**

Instance Data Block
Instance Data Header
Cyclic Data Block
Event Data Block
Service Data Block

**6-41.5.3.2.1 Instance Data Header**

The Instance Data Header contains critical axis configuration information needed to parse and apply the data contained in the three data channels. This header is only included in the Variable Connection format to accommodate multi-axis applications. Information within the header can be used by the drive communications interface to copy the individual data blocks into separate fixed memory locations for processing.

If configured for a Fixed Connection format, only the Cyclic Data Block for a single axis instance is supported so there is no need for any information on instance number or block sizing. Hence, the Instance Data Header is not included in the connection structure.

**Table 6-41.28 Instance Data Header**

Instance Data Header			
Instance Num	-	Instance Block Size	Cyclic Block Size
Cyclic Actual Block Size	Cyclic Read Block Size	Event Block Size	Service Block Size

- **Instance Number:** Same as Controller-to-Device definition
- **Instance Block Size:** Same as Controller-to-Device definition.
- **Cyclic Block Size:** Same as Controller-to-Device definition.
- **Cyclic Actual Block Size:** Same as Controller-to-Device definition.
- **Cyclic Read Block Size:** Same as Controller-to-Device definition.
- **Event Block Size:** Same as Controller-to-Device definition.
- **Service Block Size:** Same as Controller-to-Device definition.

**6-41.5.3.2.2 Cyclic Data Block**

The Cyclic Data Header at the top of the Cyclic Data Block of the Device-to-Controller Connection is always included regardless of the connection format. This header contains key elements related to the content of the Cyclic Data Block and the context of the data within the block with respect to the motion axis device. Most of these elements are established by, and are therefore direct copies of, corresponding elements of the previous Controller-to-Device Connection Cyclic Data Block. Thus, the content of the Cyclic Data Block for the Device-to-Controller Connection is ultimately determined by the motion controller.

**Table 6-41.29 Cyclic Data Block**

Cyclic Data Block			
Control Mode	Feedback Config	Axis Response	Response Status
-	Actual Data Set	Status Data Set	Axis State
Cyclic Command Data			
Cyclic Actual Data			

CIP Motion Drive, Type: 25<sub>hex</sub>

## Cyclic Data Block

## Cyclic Status Data

- **Control Mode:** Same as Controller-to-Device definition.
- **Feedback Config:** Same as Controller-to-Device definition.
- **Axis Response:** The 8-bit Device Response is an enumerated value that is used for handshaking with the corresponding Device Control element of the Controller-to-Device Connection to directly initiate motion axis operations that do not require a CIP service request. Valid Acknowledge Codes match the corresponding Request Codes of the Axis Control element, and are shown below:

Table 6-41.30 Axis Response

Acknowledge Code	Axis Response
0	No Acknowledge
1	Enable Acknowledge
2	Disable Acknowledge
3	Shutdown Acknowledge
4	Shutdown Reset Acknowledge
5	Abort Acknowledge
6	Fault Reset Acknowledge
7-127	Reserved
128-255	Vendor Specific

This Axis Control/Axis Response mechanism for initiating state changes is fully described in the State Control section of the Motion Axis Object.

- **Response Status:** When there is a non-zero Acknowledge Code in the Axis Response, a Response Status value is also provided to indicate success or failure of the requested Axis Control operation. A Response Status of 0 indicates success, while a non-zero value indicates an error. The Response Status values comply with CIP general Status Codes.
- **Command Data Set:** Same as Controller-to-Device definition.
- **Actual Data Set:** Same as Controller-to-Device definition.
- **Status Data Configuration:** Same as Controller-to-Device definition.
- **Axis State:** This data element contains the enumerated Axis State value indicating the current state of this motion axis instance according to the Motion Axis Object State Model.
- **Cyclic Actual/Status Data:** The Cyclic Actual/Status Data contains high priority data that needs to be applied to the associated motion axis instance during the next motion axis device update. This block consists of actual data elements and status data elements that are consumed by the motion controller as explicitly determined by the Actual Data Set and the Status Data Set elements in the Cyclic Data Header. See Controller-to-Device definition for details of Cyclic Actual/Status Data structure.

### 6-41.5.3.2.3 Cyclic Read Data Block

The Cyclic Read Data Block can be used to synchronously update one or more targeted Controller Axis Object attributes within the motion controller based on the current value of associated attributes in the motion axis device. This mechanism can be used in conjunction with, for example an IEC-1131based Function Block program, to implement sophisticated outer loop control based on a wide variety of available motion axis control signals. Unlike service channel Get Axis Attribute service requests, which may take several axis update cycles to process, the Cyclic Read Data mechanism guarantees the targeted parameter is updated every connection cycle.

The Cyclic Read Data Block is only supported in the Variable Connection format.

**Table 6-41.31 Cyclic Read Data Block**

Cyclic Read Data Block	
Cyclic Read Block ID	-
Cyclic Read Data	

The associated header for this block contains key elements related to the content of the Cyclic Write Data Block.

- **Cyclic Read Block ID:** This 16-bit ID determines the pre-defined Cyclic Read Block structure to apply to the Cyclic Read Data for this update. Cyclic Read Block structures are defined using the Set Cyclic Read Data List service. A successful response to this service includes a new Cyclic Read Block ID that can be used in the next connection update to pass cyclic data in this format.
- **Cyclic Read Data:** The Cyclic Read Data contains high priority data that needs to be applied to the associated controller axis instance. This block consists of signal and status data elements that are to be scaled and applied to corresponding Motion Axis Object attributes. The contents of the Cyclic Read Data are explicitly determined by the structure identified by the Cyclic Read Block ID found in the Cyclic Read Data Header.

### 6-41.5.3.3 Event Data Block

The Event Data Block allows multiple event notifications to be sent to the motion controller in a given update. Each event notification requires a specific event acknowledge indicating success or an error. The event notification data persists in the Device-to-Controller Connection data structure until the drive receives the corresponding event acknowledgement from the motion controller.

The Event Data Block for the Device-to-Controller Connection has the following format.

**Table 6-41.32 Event Data Block**

Event Data Block			
Event Checking Status			
Event ID	Event Status	Event Type	-
Event Position			
Event Time Stamp			
...			

CIP Motion Drive, Type: 25<sub>hex</sub>

- **Event Checking Status:** This 32-bit word is a bit mapped parameter used to indicate if the motion axis device is currently checking for events based on various device inputs, e.g. marker, home, and registration inputs. Event checking is initiated when the corresponding Event Checking Control bit is set in the Controller-to-Device connection. When an event occurs, the motion axis device captures both the time and exact axis position and passes this information to the motion controller in event data blocks. But for the controller to process the event data, the corresponding event checking status bit must be set. For more detail on how this word is used in event operations, refer to the event mechanism sequence in the Motion Axis object model in Chapter 5. A complete definition for the Event Checking Status attribute can be found in the Motion Axis object definition in Chapter 5.
- **Event ID:** Transaction number assigned to this event by the original event notification. Each event is assigned a new Event ID by incrementing the current Event ID stored in the motion axis device. Using the Event ID, the device is able to match the event acknowledgement to the appropriate event notification to complete the event data transaction.
- **Event Status:** Enumerated value indicating motion controller response to the event. A value of 0 indicates that the event was successfully processed. A non-zero value indicates that an error occurred in the event processing and the event must be resent.
- **Event Type:** This enumerated value describes the type of event that occurred. Valid event types are as follows:

**Table 6-41.33 Event Type**

Event Type	Event Description
0	Registration 1 Positive Edge
1	Registration 1 Negative Edge
2	Registration 2 Positive Edge
3	Registration 2 Negative Edge
4	Marker Positive Edge
5	Marker Negative Edge
6	Home Switch Positive Edge
7	Home Switch Negative Edge
8	Home Switch-Marker ++
9	Home Switch-Marker +-
10	Home Switch-Marker -+
11	Home Switch-Marker --
12-127	Reserved
128-256	Vendor specific

- **Event Position:** 32-bit integer representation of the axis position when the designated event occurred.
- **Event Time Stamp:** This element represents the 64-bit System Time value when the specified event occurred. The time stamp units are nanoseconds. Taken together with motion axis device's System Time Offset value that is part of the Device-to-Controller connection header, the motion controller has all the information it needs compute the absolute System Time when the event occurred

#### 6-41.5.3.4 Service Data Block

The service data block allows one service response per instance to be sent to the motion controller in a given update. Each service request requires a specific service response from the motion axis device indicating success or an error. In some cases the response service contains requested data. In any case, the service response data persists in the Device-to-Controller Connection data structure until the device sees the associated service request removed from the Controller-to-Device connection instance data block (Service Block Size = 0) or a new service request is issued by the controller (incremented Transaction ID).

Each service response is represented by a block of data organized as shown below.

**Design Note:** Like the request structure, the structure of the service response does not follow the standard CIP explicit messaging format. That is primarily because this connection structure is, fundamentally, a CIP Implicit I/O connection, not an Explicit Messaging connection. However, the case of a Fixed Connection format, the Service Specific Request Data defined below is sent via an Explicit Messaging connection and follows the CIP rules for explicit service request format.

**Table 6-41.34 Service Data Block**

Service Data Block			
Transaction ID	Service Code	General Status	Extended Status
Service Specific Response Data			

- **Transaction ID:** Transaction number assigned to this service response derived from the Transaction ID of the original request. Each service request is assigned a new Transaction ID by incrementing the current Transaction ID stored in the motion controller. Using the Transaction ID in the response, the controller is able to match the service response to the appropriate service request.
- **Service Code:** Identifier that determines the specific service response that follows which should match the Service code of the originating service request. A list of valid Service Codes for the Motion Axis Object is given in the Controller-to-Device section.
- **General Status:** The General Status value is provided to indicate success or failure of the requested service request. A General Status of 0 indicates success, while a non-zero value indicates an error. The General Status values follow the CIP standard for General Status codes.
- **Extended General Status:** The Extended General Status provides a method for defining vendor specific or service specific error codes. There is currently no standard definition for these codes.
- **Service Specific Response Data:** The format and syntax of the Service Specific Response Data depends on the specified Service Code.

#### 6-41.5.4 Fixed Drive Connection Format

By specifying a Fixed Connection Format, the Motion I/O Connection can be reduced to a size that is readily applicable to other CIP Networks like DeviceNet and ControlNet. In the context of a DeviceNet or ControlNet network the following features have been removed from the connection structure to support the requirements of a fixed connection size and limited network bandwidth.

- Time Stamping
- Node Faults/Alarms
- One Instance Only
- Dynamic Block Sizing
- Cyclic Read/Write Data Block
- Event Data Block
- Service Data Block

With Fixed Connection Format, service requests to the Motion Axis Object are supported only as an Explicit Messaging service.

Below is an illustrative example of the Fixed Connection Format being used in a simple variable speed drive application requiring only a velocity command and returning actual velocity. In this case, the connection size has been reduced to 16-bytes, a size that is well suited for a network like DeviceNet.

**Table 6-41.35 Fixed Controller to Device Connection Format (fixed size = 16 bytes)**

Controller to Device Connection Structure			
Connection Format	Format Revision	Update ID	Node Control
Control Mode	Feedback Configuration	Axis Control	-
Command Data Set	Actual Data Set	Status Data Set	-
Command Velocity			

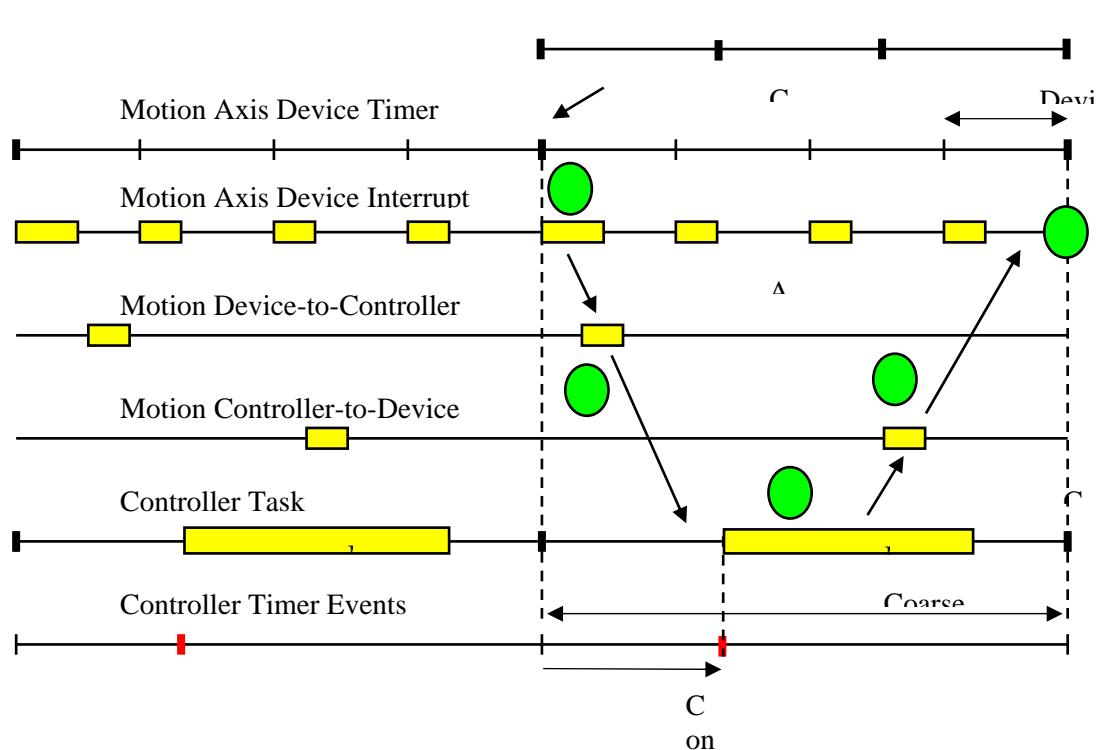
**Table 6-41.36 Fixed Device to Controller Connection Format (fixed size = 16 bytes)**

Device to Controller Connection Structure			
Connection Format	Format Revision	Update ID	Node Status
Control Mode	Feedback Configuration	Axis Response	Response Status
-	Actual Data Set	Status Data Set	Axis State
Actual Velocity			

#### 6-41.5.5 Motion Connection Timing

The general timing model for motion I/O connection data exchange is described in this following section. Data exchange between the drive and the controller is paced by the controller with one Device-to-Controller data packet sent for every Controller-to-Device data packet received. The Controller-to-Device connection packets are sent periodically according to the configured Controller Update Period. The Device Update Period, i.e. the update period at which the drive performs its control calculations, is typically much faster than the Controller Update Period. The basic motion connection timing model is illustrated in the following diagram:

Figure 6-41.4 Motion Connection Timing Model



The basic motion connection sequence consists of 5 basic steps, which are shown in the diagram and described as follows:

1. A Motion Axis Device Interrupt Service begins in response to a periodic Motion Axis Device Timer Event and the drive determines that a Coarse Update transmission to the motion controller is required. The motion axis device computes the Actual Position value (assuming Position Control Mode in this example) based on feedback count value captured in synchrony with the interrupt service timer event.
2. Once the Actual Position Input Data is loaded into the Device-to-Controller connection structure, the motion axis device transmits the data to the motion controller. The Input Data packet traverses the network and arrives at the motion controller prior to the start of the phase delayed Controller Task.
3. After a predetermined Phase Offset time from the start of last Controller Timer Event, which is typically set to be around 1/3 of the Coarse Update Period, the Controller Task starts. The task begins by reading the Actual Position Input Data from motion axis device. The motion controller then runs a Motion Planner to compute a new Command Position value to send back to the motion axis device. When gearing or camming operations are active it may be necessary to use the Actual Position of a master axis as input to compute the Command Position of one or more slave axes.

**CIP Motion Drive, Type: 25<sub>hex</sub>**

4. Once the Command Position Output Data is loaded into the Controller-to-Device Connection structure, the controller transmits the data to the motion axis device. The Output Data packet traverses the network and arrives at the motion axis device prior to the start of the next Coarse Update Period.
5. The next Drive Interrupt Service begins and the motion axis device determines that it has received fresh Output Data from the motion controller. The motion axis device then reads the Command Position Data and applies the data along with the time stamp to the device's fine interpolator/extrapolator.

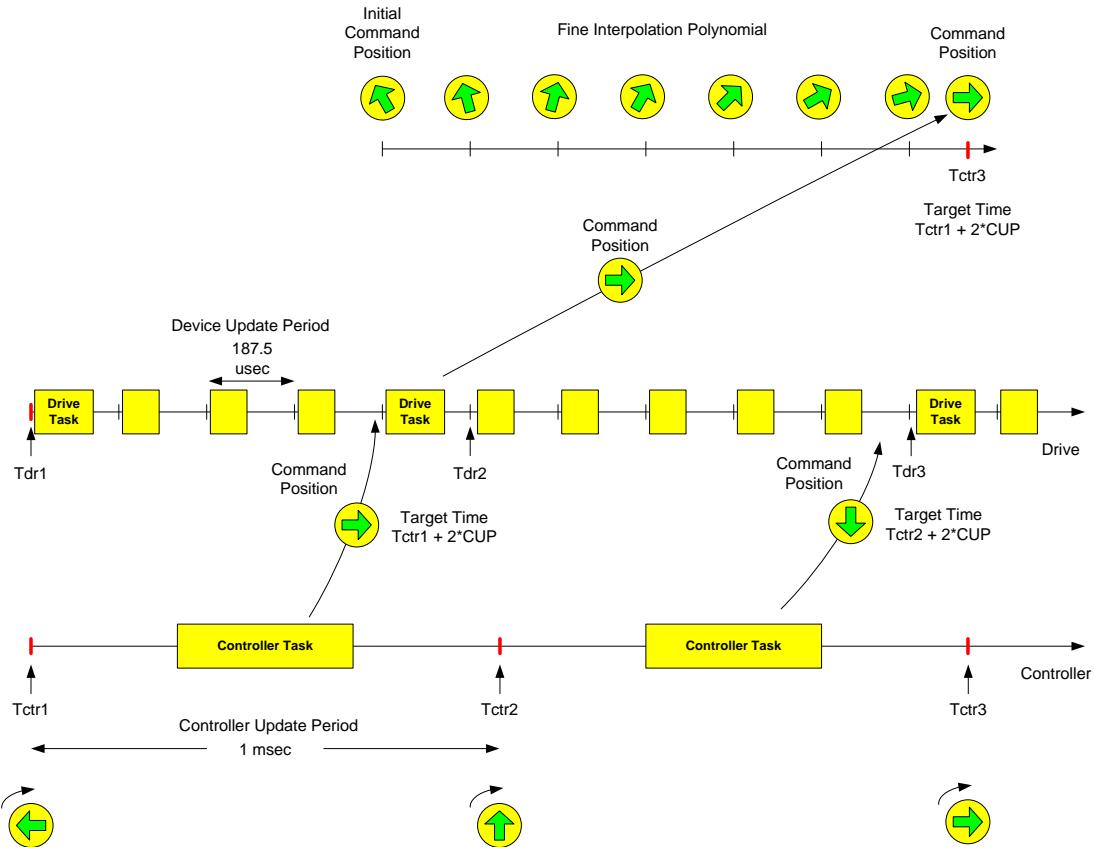
While other timing models are possible based on the motion controller configuration, all applicable motion timing models begin with the motion axis device sending actual data to the motion controller at the beginning of the Controller Update Period and end with command data arriving before the next Controller Update Period. However, it is not required that the command data sent to the motion axis device in a given update period be the result of calculations performed during that period.

Most motion control protocols require the Controller Update Period to be an integer multiple of the Device Update Period. But because the Motion I/O Connection packet includes a Time Stamp, the update period of the controller does not need to have any fixed relationship with the update period of the motion axis device.

#### **6-41.5.5.1      Controller-to-Device Connection Timing**

The Motion I/O Connection data exchange is initiated by the motion controller via the Controller-to-Device Connection packet. The inclusion of Time Stamp and Time Offset information along with the command data in this packet relieves the stringent timing requirements imposed by other motion control network protocols. The following diagram illustrates how command data and time stamps delivered by the Controller-to-Device Connection are applied to the motion axis when fine interpolation is required.

Figure 6-41.5 Controller-to-Device Connection Timing with Fine Interpolation



The following steps describe in detail how connection data is transferred from the controller to the device for fine interpolation during a typical connection cycle in the general case where the Controller Update Period (CUP) is not an integer multiple of the Device Update Period.

1. Controller Transmit: As part of the Control Task, the motion controller initiates transmission of a Controller-to-Device Connection packet with new command data to the targeted motion axis device with an incremented Update ID and a new Controller System Time Stamp and Controller System Time Offset referencing the system time at the start of the current Controller Update Period. The instance data block for the targeted axis also contains the Command Target Time, which in this example is set to +2 to account for the one Controller Update Period (CUP) transport delay and the one Controller Update Period (CUP) delay for fine interpolation.
2. Command Update ID Check: The motion axis device runs a periodic Drive Task that checks every Device Update Period for new Controller-to-Device Connection packet data. This can be easily done by checking for a changed Update ID. If the Drive Task discovers fresh data, then this is a command data update cycle and the command data must be further processed according to the following sequence.

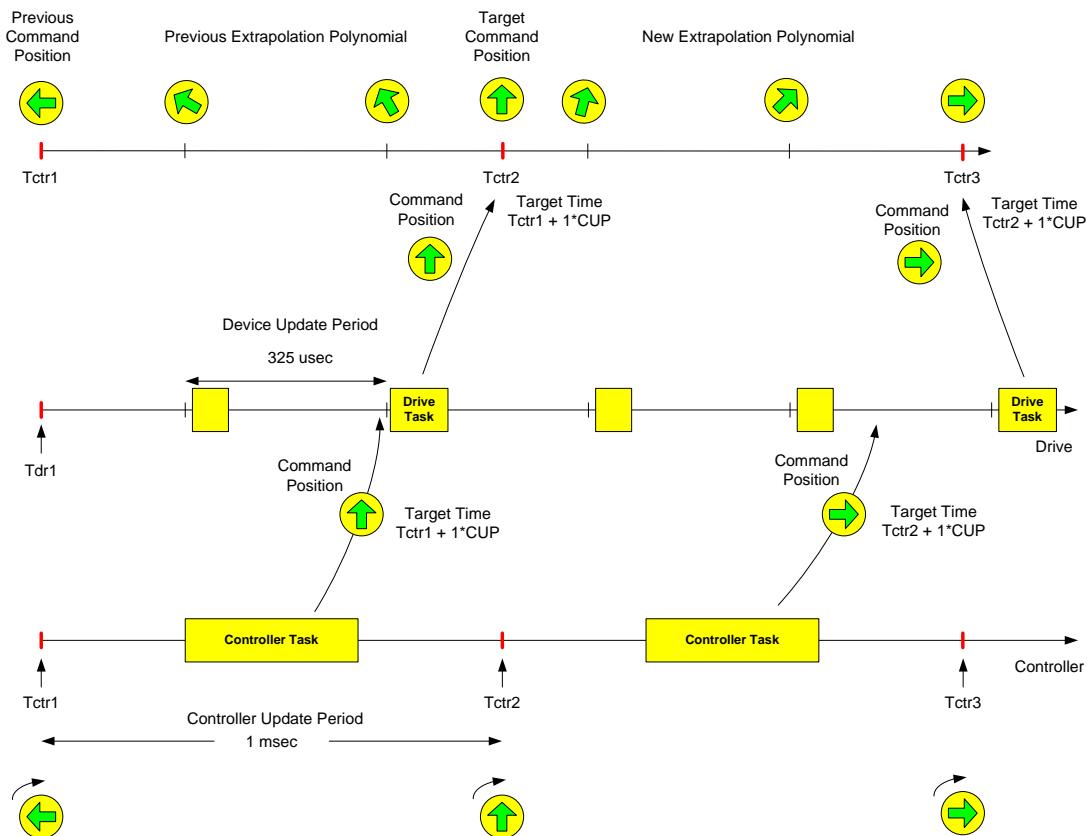
**CIP Motion Drive, Type: 25<sub>hex</sub>**

3. Synchronous Operation Check: Next, the motion axis device checks if it is synchronized. If not synchronized, skip to the Apply Command Data step since there is no need to perform the System Time Offset Check or the Late Update Check. Bypassing these checks allows for control of the motion axis device during start-up or even in the case where the device does not have any time synchronization services.
4. System Time Offset Check: Synchronous operation requires that System Time between the motion axis device and the motion controller match to make the time stamps exchanged between the devices meaningful. Since System Time can be adjusted at any time by the time master and the adjustments can propagate to the different devices in the network at different times, it is possible for System Time between the controller and the motion axis device to be skewed. This skew must be detected and, if present, the motion axis device must adjust the Controller Time Stamp to compensate for the skew. For details of this algorithm refer to the section entitled “Compensation Algorithm for Timestamps Within a Single Node” in the Time Sync Object definition in chapter 5.
5. Late Update Check: Assuming synchronous operation, the motion axis device computes the difference between the current drive update time stamp and the Controller Time Stamp passed in the Controller-to-Device Connection packet Stamp (adjusted, if necessary). If the difference is greater than Controller Update Delay High Limit \* Controller Update Period, the drive throws a Controller Update Fault. Note that if the time difference has exceeded twice the Connection Update Period, the current fine interpolator polynomial has become, effectively, an extrapolator polynomial allowing the motion axis to ride through the late data condition until the new data arrives.
6. Apply Command Data: Assuming synchronous operation and a Command Target Time of +2 for fine interpolation, the motion axis device computes coefficients for the fine interpolation polynomial based on the command reference being applied at the computed Target Time, which is the sum of the (adjusted) Controller Time Stamp (CUP), Tctr1, and the product of the Command Target Time and Controller Update Period, or in this case, 2 \* CUP. If the Target Time is less than the current System Time in the motion axis device, new coefficients to the polynomial are still computed based on this command data to improve the accuracy of the extrapolation calculations. In general, whenever command data is late, the data still represents the freshest command data available and must be applied as soon as possible. If asynchronous operation, the command data is applied to the motion axis control structure immediately.
7. Schedule Next Device-to-Controller Update: Assuming synchronous operation, the motion axis device schedules the time to send the next Device-to-Controller Connection packet to the controller. This transmission time target is the sum of the (adjusted) Controller Time Stamp, Tctrl, plus the Controller Update Period (CUP). Since, in this general case, the Controller Update Period is not an integer multiple of the Device Update Period, the transmission time target must be in the form timing window, defined as the Actual Update Window, that spans 1 Device Update Period and ends at the computed time of the next Controller Update. If asynchronous operation, the motion axis device simply proceeds to transmit the next Device-to-Controller Connection packet.
8. Follow Device-to-Controller Timing sequence in section 6-41.5.5.2.

CIP Motion Drive, Type: 25<sub>hex</sub>

If the Command Target Time is set to +1, the computed polynomial in step 6 is not applied for the purpose of fine interpolation but rather for extrapolation; the extrapolation polynomial allows the motion axis device to compute an accurate command data value at the time the device performs its control calculations based on previous axis trajectory. The diagram below illustrates this timing model in the general case where the Controller Update Period (CUP) is not an integer multiple of the Drive Update Period.

**Figure 6-41.6 Controller-to-Device Connection Timing with Unequal Update Periods**



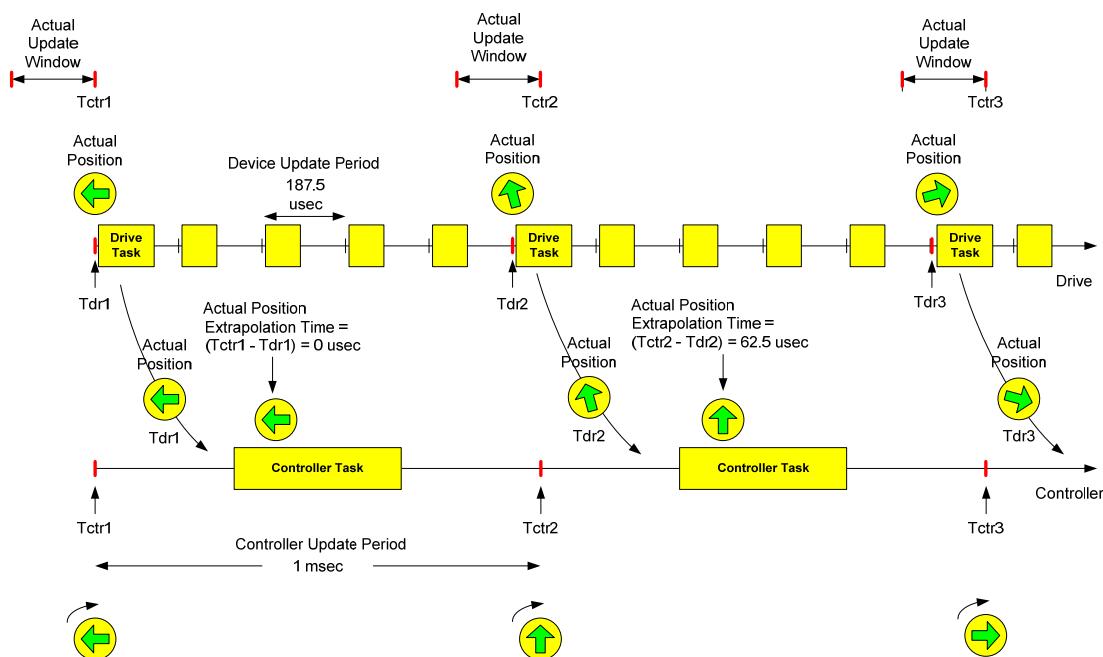
Note in the above example there are not many Device Update Periods in a given Controller Update Period. When this is the case, fine interpolation is not critical to motion axis performance and command data can be applied more directly to the motion axis' control structure without the extra delay required to support fine interpolation. Extrapolation has the disadvantage however that extrapolation error is manifested more directly to the command data resulting in rougher motion than when using fine interpolation.

All cyclic data associated with the Controller-to-Device Connection packet must be applied in the Drive Task command update to make the earliest possible use of fresh command data, computing new interpolation/extrapolation polynomial coefficients.

### 6-41.5.5.2 Device-to-Controller Connection Timing

When in synchronous mode, the Motion Device-to-Controller Connection includes a Device Time Stamp (and Device Time Offset) with the actual data to allow the motion controller to determine the position of the motion axis at the time the Controller Task update occurs. Time stamping allows the motion axis device to sample feedback and compute actual data values based on its own Device Update Period that, unlike other motion control network protocols, does not need to be strictly related to the Controller Update Period. The following diagram illustrates how actual data and time stamps delivered by the Device-to-Controller Connection are used to adjust motion axis actual position, for example, to the motion controller's timebase.

**Figure 6-41.7 Use of Time Stamp to Adjust Actual Position to the Controller's Timebase**



The following steps describe in detail how connection data is transferred from the motion axis device to the motion controller during a typical connection cycle in the general case where the Controller Update Period (CUP) is not an integer multiple of the Drive Update Period. This sequence of steps begins where the Controller-to-Device description ends as detailed in the previous section.

**CIP Motion Drive, Type: 25<sub>hex</sub>**

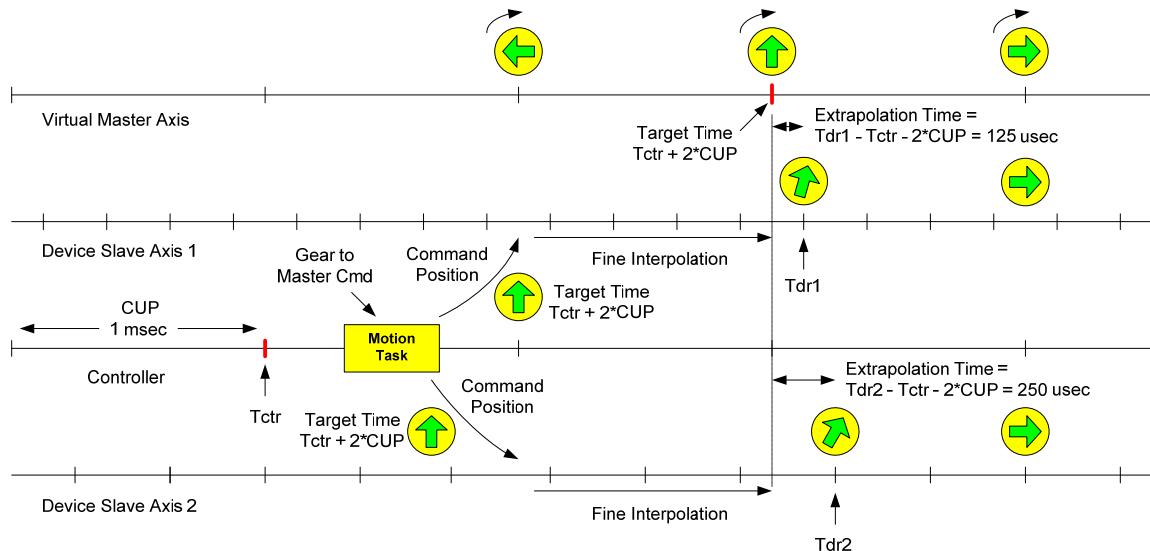
1. Actual Update Window Check: If the axis is synchronized, the motion axis device compares current Drive Task time stamp with the Actual Update Window that is determined during the last command data update. The Actual Update Window has duration of 1 Device Update Period and ends at the computed time of the next Controller Update. If the Drive Task time stamp is within the time window, this is an actual data update cycle. If the time stamp is before the window, then the motion axis device must wait for a subsequent Drive Task to send the actual data to the controller. (This prevents a condition where there is excessive time between the feedback capture and the start of the next Controller Task.) If the axis is not synchronized and it has just received a command update via the Controller-to-Device Connection, then this is also an actual update cycle, so move on to the Drive Transmit step.
2. Drive Transmit: If this is an actual update cycle, then the motion axis device sends the Device-to-Controller Connection packet to the motion controller with the latest Actual Data from this Drive Task, including, if synchronized, the current Device Time Stamp (and Device Time Offset), and an incremented Update ID. If not synchronized, just send the Update ID of the preceding Controller-to-Device Connection packet. All additional data sent to the motion controller in this packet may be derived from the previous Drive Task. This allows the drive transmission to occur at the earliest point in the Drive Task execution.
3. Actual Update ID Check: In the next Controller Task, the motion controller checks for new data from the drive by checking for a changed Update ID. The following steps are performed regardless of whether or not the Update ID has changed. Note that the Update ID is the only way to detect for new actual data when the motion axis device is not synchronized.
4. Sync Flag Check: Motion axis device checks the Sync Mode bit of the Drive Node Status byte to determine if the motion axis is synchronized. If not, skip to the Apply Actual Data step to avoid Late Update checking and Time-Stamp Correction. Bypassing these subsequent steps allows the motion axis to operate during start-up or even in the case where the motion axis device does not have any time synchronization services.
5. System Time Offset Check: Synchronous operation requires that System Time between the motion axis device and the motion controller to match to make the time stamps exchanged between the devices meaningful. Since System Time can be adjusted at any time by the time master and the adjustments can propagate to the different devices in the network at different times, it is possible for System Time between the controller and the motion axis device to be skewed. This skew must be detected and if present, the motion axis device must adjust the Device Time Stamp to compensate for the skew. For details of this algorithm refer to the section entitled “Compensation Algorithm for Timestamps Within a Single Node” in the Time Sync Object definition in chapter 5.
6. Late Update Check: The motion controller computes the difference between the current Connection Update Period time stamp and the Time Stamp in the Device-to-Controller Connection packet. If the difference is greater than Missed Update Tolerance \* Update Period, the controller throws a Controller Sync Fault.

7. Time-Stamp Correction: If the previously computed time difference is non-zero, then extrapolate the actual data value based on previous axis actual trajectory to line up with the controller's time stamp. This correction is necessary because the motion planner assumes that actual input data is implicitly time stamped to the beginning of the Controller Update Period.
8. Apply Actual Data: Controller applies actual data as inputs to the motion planner, which computes new command reference data for the next Controller-to-Device update.

#### 6-41.5.5.3 Device Update Period Independence

The timing diagram below illustrates how two motion axes can be tightly coordinated despite having different Device Update Periods and despite having an associated Controller Update Period that is not an integer multiple of either Device Update Period.

**Figure 6-41.8 Coordination of Two Drives with Different Update Periods.**



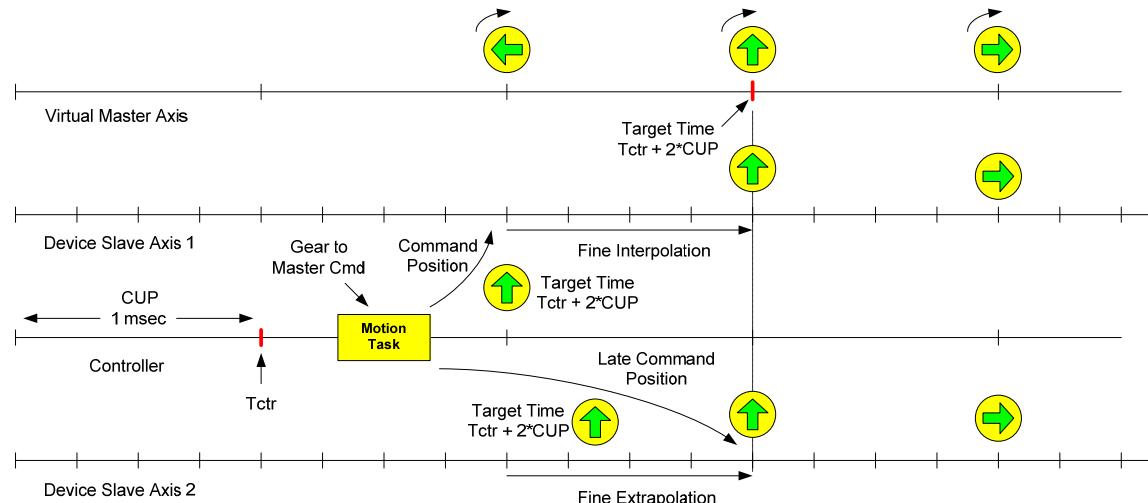
**CIP Motion Drive, Type: 25<sub>hex</sub>**

In the above timing diagram, the controller's motion planner task sends identical command positions and time stamps to two slave motion axes that, while synchronized with System Time, are running at different motion axis update rates. When the command position data arrives at the two devices, they use the Controller Time Stamp, the Command Target Time, and the Controller Update Period to compute new coefficients to the interpolation polynomial based on the constraint that the polynomial value at time equal to (Controller Time Stamp + Command Target Time \* Controller Update Period) is the specified Command Position value. Since there is no dependency on the motion axis update rate, the polynomial coefficients computed by each device are identical. Since neither device has an update that coincides with this target time, they use the fine interpolation polynomial to calculate the command position reference for each motion axis update until a fresh command position is received from the motion controller. If a new command position does not arrive until well after the target time, the motion axis device continues to use the same polynomial equation to "extrapolate" command position for subsequent motion axis updates as shown in the above diagram. This extrapolation continues until fresh data arrives and new coefficients can be calculated. In this way, whether by interpolation or extrapolation, each slave axis runs smoothly and the two axes stay phase locked with the master axis.

**6-41.5.5.4 Transmission Latency Independence**

Precise coordination of multiple motion axes can be maintained even when the Controller-to-Device Connection packets incur significant delays while traveling across the CIP network. In the diagram below, the packet for Slave Drive Axis 2 has incurred a significant delay during transmission. As a result, the command position for this axis must be extrapolated from the last fine interpolation polynomial. This allows the axis to move smoothly through a transmission latency disturbance. When the new command data does arrive, the new command value may not agree with extrapolated value due to extrapolation error. This error can result in a disturbance to the motion profile. The magnitude of the extrapolation error depends on the dynamics of the motion profile and the controller update rate. In most real-world applications, transmission latencies lasting several update periods can occur without any noticeable disturbance to the associated motion profile.

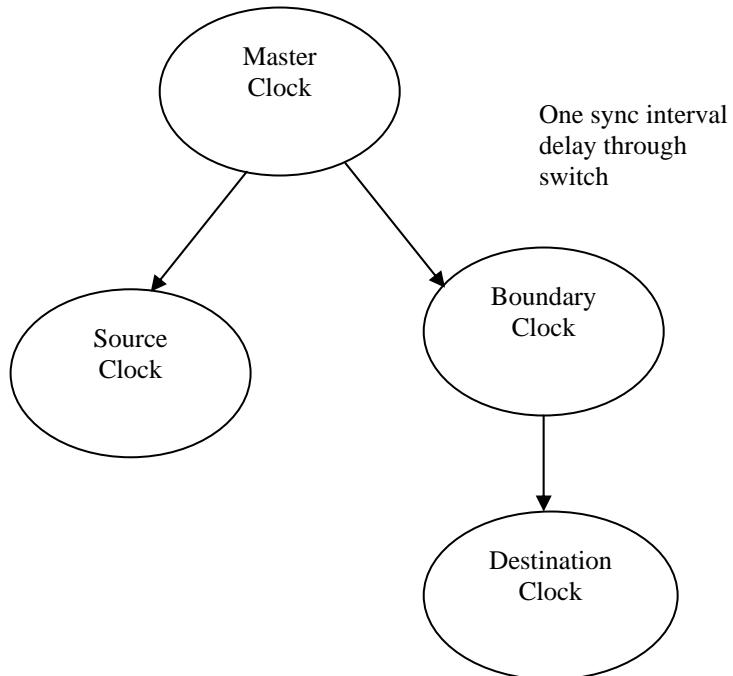
**Figure 6-41.9 Coordination of Multiple Drive Axes in case of Delayed Controller-to-Drive Connection Packets**



### 6-41.5.5.5 System Time Offset Compensation

Motion control on CIP is built on the foundation of CIP Sync™ and its underlying IEEE-1588 time synchronization protocol, which defines a mechanism to distribute and synchronize time for devices on a network. Unfortunately, IEEE-1588 makes no guarantee as to how soon all the devices in the system respond to a step change in the master's time and has no provision for handling this condition. This step change in time may be caused, for example, by the user manually adjusting the master's clock due to correct drift in the clock or by a change in time mastership. According to IEEE-1588, step changes to the master clock time must propagate through the distributed time system, and the time to propagate this step change in time through the system from the time master to the furthest time slave is a function of the sync packet interval (default is 2 seconds) and the number of hops (1 hop per 1588 boundary clock, e.g. a switch implementing the 1588 protocol). This is illustrated in the figure below.

**Figure 6-41.10 Propagation of a Step Change in Time**



In the illustration, the Source Clock is going to receive a System Time adjustment one sync interval before the Destination Clock. Thus, System Time for the Source Clock is going to be skewed relative to System Time for the Destination Clock for an entire sync packet interval.

The variability in propagating this time adjustment through the system creates a problem for time stamped based motion control in that the time stamp in the source device, say the controller, may not be applicable when received by the destination device, say a drive, because System Time for the source and destination devices is skewed. This effect is further illustrated in the following table.

**CIP Motion Drive, Type: 25<sub>hex</sub>****Table 6-41.37 Propagation of a Step Change in Time**

Time Sync Interval	Master Time	Source Time (Controller)	Destination Time (Drive)
1	100	100	100
2	200 + 1000	200	200
3	1300	300 + 1000	300
4	1400	1400	400 + 1000
5	1500	1500	1500

At Time Sync Intervals 1 and 2, both the timestamp source clock in the motion controller, and timestamp destination clock in the motion axis device both have the same notion of time, i.e. System Time. At Sync Interval 2, a step change in time occurs at the master clock that propagates to the motion controller in Sync Interval 3, thus affecting the source clock time stamps sent to the motion axis device. It is not until Sync Interval 4 that the motion axis device sees the step change. Any time stamp data sent from the controller to the motion axis device between Sync Intervals 3 and 4 will not correlate with its notion of time.

CIP Motion and CIP Sync extend the 1588 protocol to handle this condition. This is done by defining a run-time algorithm in the timestamp destination device that can detect this time step condition and adjust the value of the received timestamp so that it is accurate. Two conditions are possible:

1. The source device has seen a step change in time but the destination device has not.
2. The destination device has seen a step change in time but the source device has not.

The key component of the algorithm is the System Time Offset value defined as part of the CIP Time Sync Object. This offset value is added to the local clock time to generate System Time, that is:

$$\text{System Time} = \text{Local Clock} + \text{System Time Offset}$$

When the system is synchronized, at a given moment of time, the Local Clock values and the System Time Offset values may differ from device to device, but the System Time value should be the same within the accuracy of the 1588 protocol implementation.

A step change in time is indicated by a change in the System Time Offset value of either the source or destination devices. The source's System Time Offset is sent to the destination device as an offset along with the source's System Time as the data time stamp. The destination device compares the offset received to the previously received offset to determine if a step change has occurred and adjusts the received timestamp value accordingly.

The algorithm is stated as follows:

$\text{Timestampcomp} = \text{TimestampRec} +$   
 $((\text{Offset}_{\text{dest}} - \text{Offset}_{\text{dest}(\text{last})}) - (\text{Offset}_{\text{src}} - \text{Offset}_{\text{src}(\text{last})}))$   
 If  $|((\text{Offset}_{\text{dest}} - \text{Offset}_{\text{dest}(\text{last})}) - (\text{Offset}_{\text{src}} - \text{Offset}_{\text{src}(\text{last})}))| \leq \text{Sync Threshold}$   
 $\text{Offset}_{\text{dest}(\text{last})} = \text{Offset}_{\text{dest}}$   
 $\text{Offset}_{\text{src}(\text{last})} = \text{Offset}_{\text{src}}$

Where:

**CIP Motion Drive, Type: 25<sub>hex</sub>**

TimestampRec = received time stamp.

Offsetdest = the current value of the System Time Offset at the destination.

Offsetdest(last) = the previous value of the System Time Offset at the destination.

Offsetsrc = the received value of the System Time Offset from the source.

Offsetsrc(last) = the previous value of the System Time Offset from the source.

Sync Threshold = a number that defines that synchronization tolerance for the device application. For a typical vector drive this number should be on the order of 1 microsecond.

The following table provides an example of how System Time Offset Compensation works. For illustration purposes, a packet is assumed to be sent at the sync interval with no propagation delay.

**Table 6-41.38 Propagation of a Step Change in Time**

Time Sync Interval	Master Time	Timestamp Source (Controller)					Timestamp Destination (Motion Axis)				
		Local Clock	Offset	Last Offset	Local Time	Time stamp	Local Clock	Offset	Last Offset	Local Time	Adj. Time stamp
1	100	0	100	100	100	100	0	100	100	100	100
2	200+ 1000	100	100	100	200	100	100	100	100	200	200
3	300	200	100 + 1000	100	1300	1300	200	100	100	300	300
4	1400	300	1100	100	1400	1400	300	100 + 1000	100	1400	1400
5	1500	400	1100	1100	1500	1500	400	1100	1100	1500	1500

At Sync Interval 2 a step change of 1000 occurs in the master clock and is propagated to the timestamp source clock, say the controller, in Sync Interval 3. The time stamp received at the destination clock, the motion axis device in this example, is compensated for by 1000 according to the above algorithm that results in an Adjusted Controller Time Stamp value that is consistent with the device's Local Time even though it has not yet seen the step change. In Sync Interval 4, the device sees the step change but no adjustment is made to the time stamp because in the compensation formula above the source and destination delta offset terms cancel out and the Last Offset values are updated as shown in sync interval 5 and no further offset compensation is required.

## 6-41.6 Drive Startup Procedure

In this section the start-up process of a motion axis device is discussed. For the sake of clarity the exact messages and their formats will not be discussed. The objective is to discuss the steps and the exchanges that take place between the controller and the drive in order to correctly configure the motion axis device.

The drive startup involves three distinct processes, which are initiated by the CIP Controller and responded to by each motion axis device on the network. They are:

1. Device Connection Creation.
2. Device Object Configuration.
3. Time Synchronization

These processes may occur in parallel or consecutively depending on the particular implementation. Device Configuration may occur via connected or unconnected explicit messaging. The startup process is outlined below.

### 6-41.6.1 Motion I/O Connection Creation

Motion I/O Connections are created on EtherNet/IP using either the Forward\_Open or Large\_Forward\_Open services. These services should specify the maximum size of the Control-to-Device and Device-to-Control Connection Data Structures applied to the Motion Axis Object. Configuration of the produced and consumed I/O connection paths is as described above.

### 6-41.6.2 Motion Axis Object Configuration

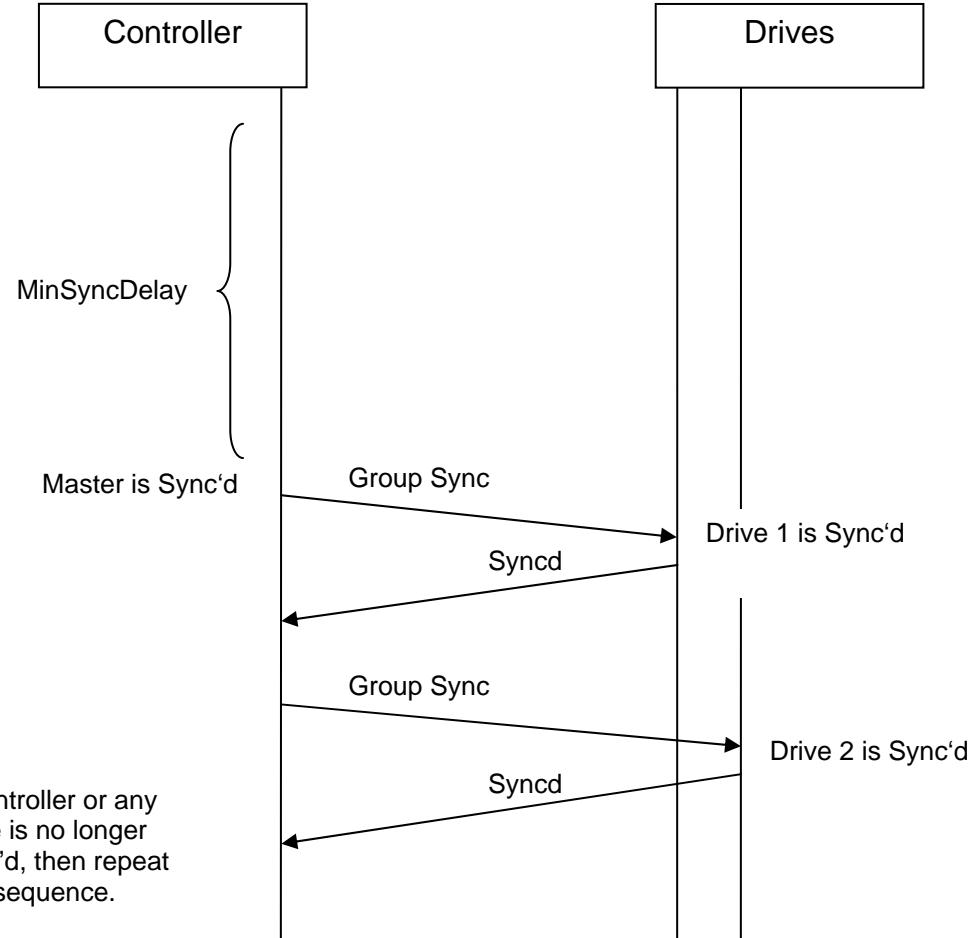
Motion Axis Object configuration is accomplished using either the Set\_Attribute\_List service or the object specific Set Drive Attribute List service defined by the Motion Axis Object and supported by the Motion I/O Connection.

### 6-41.6.3 Time Synchronization

All motion axis devices that support time synchronization services are required to be synchronized with a motion controller as part of a group, commonly embodied as a Motion Group. All devices in this group are sent a GroupSync service that contains the Controller Update Period and the System Time Offset of the motion controller. The objective of this procedure is to guarantee that all the devices are synchronized or, in other words, that they have the same notion of time. This is accomplished by guaranteeing that no step changes to the motion axis device's System Time Offset value has occurred recently and the motion controller's System Time Offset and motion axis device's System Time Offset values are based on a System Time reference that is consistent between the controller and the motion axis device.

The following diagram illustrates how the motion controller first waits until its local clock is synchronized to the master System Time clock and then waits a minimum delay period to make sure there are no step changes to the System Time Offset. By step changes we mean changes to the System Time Offset that are more than a predefined threshold defined by the Sync Threshold attribute

Figure 6-41.11 Group Sync of CIP Motion Drives



If there have been no System Time Offset steps during this period, the motion controller is considered to be synchronized and initiates a series of GroupSync services to each of the motion axis devices. Each device returns a response indicating whether it is also synchronized according to the same criterion described for the motion controller. If the controller and all motion axis devices indicate they are synchronized, then the Motion Group is considered to be synchronized.

There are two device specific parameters that are used by a motion device to determine if the device is group synchronized.

1. Sync Update Delay
2. Sync Threshold

The Sync Update Delay is calculated by the Motion Axis Object as the minimum time to wait after the last System Time Offset change greater than Sync Threshold before the device is considered to be group synchronized. This value is on the product of the number of networks hops the motion axis device is away from the grandmaster clock and the time synchronization interval. The Sync Update Delay attribute is typically on the order of seconds.

**CIP Motion Drive, Type: 25<sub>hex</sub>**

The Sync Threshold is a class attribute of the Motion Axis Object whose value is used to determine if the device's local clock is frequency synchronized to the master System Time clock. Sync Threshold depends on the IEEE-1588 implementation, but is typically on the order of microseconds for a hardware assisted clock implementation. Offset time changes greater than the Sync Threshold are considered step changes to the System Time Offset. Offset time changes less than the Sync Threshold are considered normal corrections to System Time to compensate for clock drift and network latency variation.

Once the motion controller and all the motion axis devices in the Motion Group are group synchronized, the controller can then set the Synchronous Control bit in the Controller-to-Device Connection header of the next cyclic packet sent to the motion axis devices indicating that the device can schedule its next Device-to-Controller Connection update based on the Controller Time Stamp and Controller Update Period contained in the connection data. The device indicates that it is now sending scheduled connection data based on the Controller Update Period by setting the Synchronous Mode bit in the Device-to-Controller Connection header of the next cyclic packet sent to the motion controller. Data exchange between the controller and the motion axis device proceed thereafter according the Motion Timing Model.

**6-41.7****Device Visualization**

The motion axis device requires visualization components to quickly diagnose the condition of the device. These components range from bicolor LEDs to multi-character alphanumeric displays. This section defines the requirements for these visualization components and appropriate labels for these components when employed by the device. When applied to different CIP Networks, the Network Status and Module Status LED behavior shall conform to the LED behavior of the associated CIP network to which the device is connected. The mapping of the device's Module Status LED states to the Motion Axis Object states is fully defined in the Motion Axis Object section on Visualization.

**Table 6-41.39 Motion Visualization Components**

<b>Visualization Component</b>	<b>Need in Implementation</b>	<b>Associated Label</b>
Network Status LED	Required	"Net Status" "NS"
Module Status LED	Required	"Mod Status" "MS"
Drive Status LED	Conditional *	"Axis Status" "XS" "Axis # Status" "XS#***"
Seven-Segment Display	Optional	N/A
Multi-char Alphanumeric Display	Optional	N/A

\* Requirement may be waived by using a Multi-character Alphanumeric Display.

\*\* Multi-axis drive devices need multiple Drive Status LEDs, one for each drive axis.

**6-41.8****Ethernet Quality of Service (QoS)**

A motion axis device targeted for operation on EtherNet/IP shall utilize the standard EtherNet/IP prioritization scheme to be published in Volume 2. This scheme has not yet been finalized but it is likely to utilize Tagged Frames, as defined by IEEE 802.1p, which has been formally incorporated into 802.1Q. Some motion applications may require the use of Tagged Frames to achieve the required level of performance. Therefore, designers must consider the selection of hardware and firmware drivers to support "Tagged Frames" with user-selectable priority in their design.

## 6-42 Enhanced Mass Flow Controller Device

**Device Type:** 27<sub>hex</sub>

### 6-42.1 Introduction

A Mass Flow Controller is a device that measures and controls the mass flow rate of gas or liquid. It contains three principle components: a mass flow rate sensor which can be one of a variety of types, including thermal or pressure-based; a mass flow rate metering valve which can be actuated by one of a variety of actuator types, including solenoid, voice coil or piezo; and, a controller which closes the loop by receiving a setpoint and driving the actuator such that the mass flow rate is controlled to the set point.

This Enhanced Mass Flow Controller Device Type differs from Mass Flow Controller Device Type 1A<sub>hex</sub> by supporting both upstream process gas Temperature and upstream process gas Pressure in separate instances of the S-Analog Sensor Object.

### 6-42.2 Object Model

The Object Model in Figure 6-42.1 represents an Enhanced Mass Flow Controller Device. The table below indicates:

- the object classes present in this device
- whether or not the class is required
- the number of instances present in each class

**Table 6-42.1 Objects Present in an Enhanced Mass Flow Controller Device**

Object Class	Optional/Required	# of Instances
Identity	Required	1
Message Router	Required	1
DeviceNet	Required	1
Connection	Required	at least 1 I/O and 1 Explicit
Assembly	Required	at least 1 Input and 1 Output <sup>1</sup>
S-Device Supervisor	Required	1
S-Gas Calibration	Required	1 or more
S-Analog Sensor	Required	3 Instance 1 – flow Instance 2 – pressure Instance 3 - temperature
S-Analog Actuator	Conditional <sup>1</sup>	1
S-Single Stage Controller	Conditional <sup>1</sup>	1

<sup>1</sup> Required for a Mass Flow Controller, a device that contains a Valve and a Controller. Not supported in a Mass Flow Meter Device (an MFC without a Valve or a Controller).

### 6-42.3 Class Subclasses

Each class level subclass defines a unique meaning for an overlapping range of class attribute IDs and/or class service IDs. The range for the subclass definitions begins at ID 96 and numbers downward for attributes, and ID 63<sub>hex</sub> and numbers downward for services. The subclass for a given class is identified by the value of its Subclass class attribute. There are no class level subclasses specified for this device.

### 6-42.4 Instance Subclasses

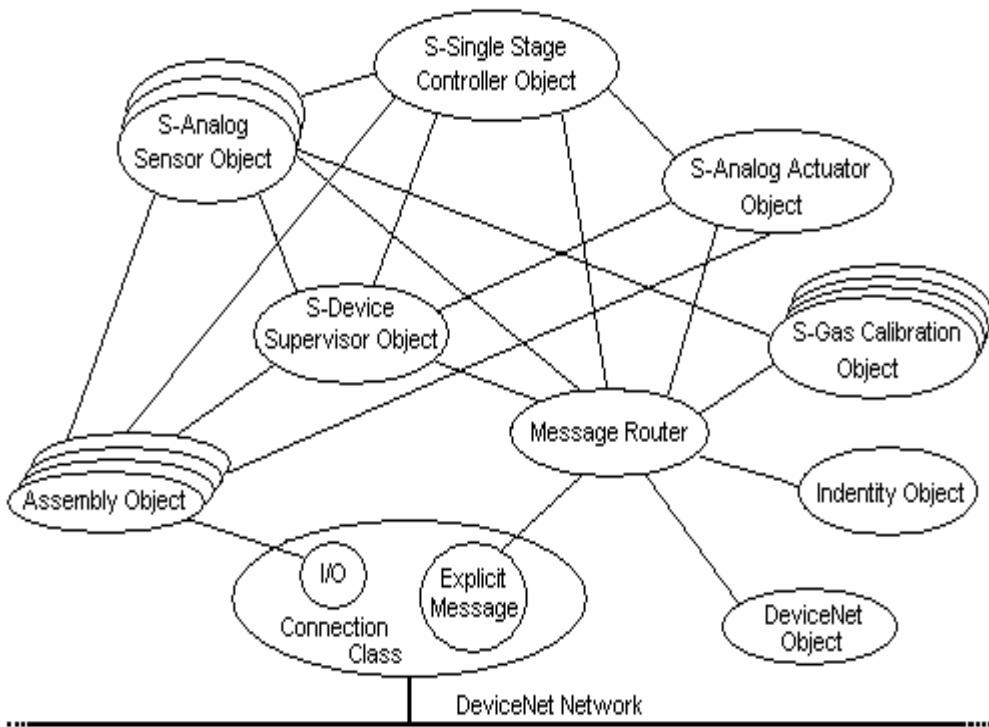
Each instance level subclass defines a unique meaning for an overlapping range of instance attribute IDs and/or instance service IDs. The range for subclass definitions begins at ID 96 and numbers downward for attributes, and ID 63<sub>hex</sub> and numbers downward for services. The subclass for a given instance is identified by the value of its Subclass instance attributes. The following tables identify which instance IDs are assigned subclasses for this device.

**Table 6-42.2 S-Analog Sensor Object Subclasses**

Instance ID	Subclass Name	Subclass ID (Attribute 99 Value)	Required	Function	Restrictions
1	Flow Diagnostics	01	Required	Added diagnostics for EMFC	None

**Table 6-42.3 S-Gas Calibration Object Subclass**

Instance ID	Subclass Name	Subclass ID (Attribute 99 Value)	Required	Function	Restrictions
1	Standard T & P	01	Optional	Standard Temperature and Pressure	None

**Figure 6-42.1 Object Model for the EMFC device**

## 6-42.5 How Objects Affect Behavior

**Table 6-42.4 Object Effect on Behavior**

Object	Effect on behavior
Identity	Supports the Reset service. Upon receipt of a <i>Reset Service Request</i> of any <i>Type</i> , the Identity Object sends a <i>Reset Service Request</i> to the S-Device Supervisor.
Message Router	No effect
DeviceNet	Configures port attributes (node address, data rate, and BOI)
Connection Class	Contains the number of logical ports into or out of the device
Assembly	Defines input/output and configuration data format
S-Device Supervisor	Supports the Stop, Start, Reset, Abort, Recover and Perform_Diagnostic services for ALL Application Objects in the device and consolidates the Exception Conditions and Application Objects' Status. This object behaves differently from the Identity Object in that the S-Device Supervisor object provides a single point of access to the Application Objects only; it does not affect the DeviceNet specific objects (i.e., Identity, DeviceNet, Connection, etc.).
S-Gas Calibration	Modifies the correction algorithm of the S-Analog Sensor object which includes the selection mechanism to enable an S-Gas Calibration object instance.
S-Analog Sensor	Feeds the process variable to the Single Stage Controller object
S-Single Stage Controller	Feeds the control variable to the Analog Actuator object
S-Analog Actuator	Operates the Flow Control Valve of the device

## 6-42.6 Defining Object Interfaces

**Table 6-42.5 Object Interfaces**

Object	Interface
Identity	Message Router
Message Router	Explicit Messaging Connection Instance
DeviceNet	Message Router
Connection Class	Message Router
Assembly	I/O Connection or Message Router
S-Device Supervisor	Assembly or Message Router
S-Gas Calibration	Message Router
S-Analog Sensor	Assembly or Message Router
S-Single Stage Controller	Assembly or Message Router
S-Analog Actuator	Assembly or Message Router

## 6-42.7 I/O Assembly Instances

The following table identifies the I/O assembly instances supported by the EMFC and EMFM devices.

**Table 6-42.6 I/O Assembly Instances**

Number	Required		Type	Name
	EMFC	EMFM		
1	N	N	Input	Flow
2	Y (default)	Y (default)	Input	Status and Flow
3	N	N	Input	Status, Flow and Valve
4	N	N	Input	Status, Flow, and Setpoint
5	N	N	Input	Status, Flow, Setpoint and Valve
6	Y	N	Input	Status, Flow, Setpoint, Override and Valve
7	Y (default)	N	Output	Setpoint
8	Y	N	Output	Override and Setpoint
9	N	N	Input	Status
10	N	N	-	(assembly not used)
11	N	N	-	(assembly not used)
12	N	N	-	(assembly not used)
13	N	N	Input	FP-Flow
14	Y	Y	Input	FP-Status and Flow
15	N	N	Input	FP-Status, Flow and Valve
16	N	N	Input	FP-Status, Flow, and Setpoint
17	N	N	Input	FP-Status, Flow, Setpoint and Valve
18	Y	N	Input	FP-Status, Flow, Setpoint, Override and Valve
19	Y	N	Output	FP-Setpoint
20	Y	N	Output	FP-Override and Setpoint
21	Y	Y	Input	Status, Flow, Pressure and Temperature
22	N	N	Input	Status, Flow, Valve, Pressure and Temperature

**Enhanced Mass Flow Controller, Type: 27<sub>hex</sub>**

Number	Required		Type	Name
	EMFC	EMFM		
23	Y	Y	Input	FP – Status, Flow, Pressure and Temperature
24	N	N	Input	FP – Status, Flow, Valve, Pressure and Temperature
25	N	N	Input	Exception Detail Alarm and Exception Detail Warning

Note that FP refers to “Floating Point”, which is analogous to the data type “REAL”.

**6-42.8 I/O Assembly Object Instance Data Attribute Format**

The manufacturer of an Enhanced Mass Flow Controller Device must specify which Assembly instances are supported by the device.

The I/O Assembly Data attribute has the format shown below.

**Table 6-42.7 I/O Assembly Data Attribute**

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	0	Flow (low byte)							
	1	Flow (high byte)							
2	0	Status							
	1	Flow (low byte)							
	2	Flow (high byte)							
3	0	Status							
	1	Flow (low byte)							
	2	Flow (high byte)							
	3	Valve (low byte)							
	4	Valve (high byte)							
4	0	Status							
	1	Flow (low byte)							
	2	Flow (high byte)							
	3	Setpoint (low byte)							
	4	Setpoint (high byte)							
5	0	Status							
	1	Flow (low byte)							
	2	Flow (high byte)							
	3	Setpoint (low byte)							
	4	Setpoint (high byte)							
	5	Valve (low byte)							
	6	Valve (high byte)							

Enhanced Mass Flow Controller, Type: 27<sub>hex</sub>

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
6	0								Status
	1								Flow (low byte)
	2								Flow (high byte)
	3								Setpoint (low byte)
	4								Setpoint (high byte)
	5								Override
	6								Valve (low byte)
	7								Valve (high byte)
7	0								Setpoint (low byte)
	1								Setpoint (high byte)
	0								Override
8	1								Setpoint (low byte)
	2								Setpoint (high byte)
	0								Status
9	13								Flow (low byte)
	0								Flow
	1								Flow
	2								Flow (high byte)
14	0								Status
	1								Flow (low byte)
	2								Flow
	3								Flow
	4								Flow (high byte)
15	0								Status
	1								Flow (low byte)
	2								Flow
	3								Flow
	4								Flow (high byte)
	5								Valve (low byte)
	6								Valve
	7								Valve
	8								Valve (high byte)
16	0								Status
	1								Flow (low byte)
	2								Flow
	3								Flow
	4								Flow (high byte)
	5								Setpoint (low byte)
	6								Setpoint
	7								Setpoint
	8								Setpoint (high byte)

Enhanced Mass Flow Controller, Type: 27<sub>hex</sub>

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
17	0	Status							
	1	Flow (low byte)							
	2	Flow							
	3	Flow							
	4	Flow (high byte)							
	5	Setpoint (low byte)							
	6	Setpoint							
	7	Setpoint							
	8	Setpoint (high byte)							
	9	Valve (low byte)							
	10	Valve							
	11	Valve							
	12	Valve (high byte)							
18	0	Status							
	1	Flow (low byte)							
	2	Flow							
	3	Flow							
	4	Flow (high byte)							
	5	Setpoint (low byte)							
	6	Setpoint							
	7	Setpoint							
	8	Setpoint (high byte)							
	9	Override							
	10	Valve (low byte)							
	11	Valve							
	12	Valve							
	13	Valve (high byte)							
19	0	Setpoint (low byte)							
	1	Setpoint							
	2	Setpoint							
	3	Setpoint (high byte)							
20	0	Override							
	1	Setpoint (low byte)							
	2	Setpoint							
	3	Setpoint							
	4	Setpoint (high byte)							

Enhanced Mass Flow Controller, Type: 27<sub>hex</sub>

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
21	0	status							
	1	Flow (low byte)							
	2	Flow (high byte)							
	3	Pressure (low byte)							
	4	Pressure (high byte)							
	5	Temperature (low byte)							
	6	Temperature (high byte)							
22	0	Status							
	1	Flow (low byte)							
	2	Flow (high byte)							
	3	Valve (low byte)							
	4	Valve (high byte)							
	5	Pressure (low byte)							
	6	Pressure (high byte)							
	7	Temperature (low byte)							
	8	Temperature (high byte)							
23	0	Status							
	1	Flow (low byte)							
	2	Flow							
	3	Flow							
	4	Flow (high byte)							
	5	Pressure (low byte)							
	6	Pressure							
	7	Pressure							
	8	Pressure (high byte)							
	9	Temperature (low byte)							
	10	Temperature							
	11	Temperature							
	12	Temperature (high byte)							

Enhanced Mass Flow Controller, Type: 27<sub>hex</sub>

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
24	0	Status							
	1	Flow (low byte)							
	2	Flow							
	3	Flow							
	4	Flow (high byte)							
	5	Valve (low byte)							
	6	Valve							
	7	Valve							
	8	Valve (high byte)							
	9	Pressure (low byte)							
	10	Pressure							
	11	Pressure							
	12	Pressure (high value)							
	13	Temperature (low byte)							
	14	Temperature							
	15	Temperature							
	16	Temperature (high byte)							
25	0	Status							
	1	Exception Detail Alarm 0 (size, common)							
	2	Exception Detail Alarm 1 (common 0)							
	3	Exception Detail Alarm 2 (common 1)							
	4	Exception Detail Alarm 3 (size, device)							
	5	Exception Detail Alarm 4 (device 0)							
	6	Exception detail Alarm 5 (device 1)							
	7	Exception Detail Alarm 6 (size, manufacturer)							
	8	Exception Detail Alarm 7 (manufacturer, 0)							
	9	Exception Detail Warning 0 (size, common)							
	10	Exception Detail Warning 1 (common 0)							
	11	Exception Detail Warning 2 (common 1)							
	12	Exception Detail Warning 3 (size, device)							
	13	Exception Detail Warning 4 (device 0)							
	14	Exception Detail Warning 5 (device 1)							
	15	Exception Detail Warning 6 (size, manufacturer)							
	16	Exception Detail Warning 7 (manufacturer, 0)							

**6-42.8.1 Mapping I/O Assembly Data Attribute Components**

Each of the S-Analog Sensor, S-Analog Actuator and S-Single Stage Controller object definitions specifies a behavior that modifies the Data Type of certain attributes based upon the first valid I/O connection established to an Assembly Object instance. In order to maintain consistency, this device type will only allow connections to either INT or REAL based Assembly instances. Once a valid connection is established, attempts to configure connections to a different type of Assembly instance will return an error

**Enhanced Mass Flow Controller, Type: 27<sub>hex</sub>**

The following table indicates the I/O assembly Data attribute mapping for this EMFC device.

**Table 6-42.8 I/O Assembly Data Mapping**

Data Component Name	Class		Instance Number	Attribute		
	Name	Number		Name	Number	Type
Flow	S-Analog Sensor	31 <sub>hex</sub>	1	Indicated Flow	6	INT
FP-Flow	S-Analog Sensor	31 <sub>hex</sub>	1	Indicated Flow	6	REAL
Valve	S-Analog Actuator	32 <sub>hex</sub>	1	Value	6	INT
FP-Valve	S-Analog Actuator	32 <sub>hex</sub>	1	Value	6	REAL
Setpoint	S-Single Stage Controller	33 <sub>hex</sub>	1	Setpoint	6	INT
FP-Setpoint	S-Single Stage Controller	33 <sub>hex</sub>	1	Setpoint	6	REAL
Status	S-Device Supervisor	30 <sub>hex</sub>	1	Exception Status	12	BYTE
Override	S-Analog Actuator	32 <sub>hex</sub>	1	Override	5	USINT
Exception Detail Alarm	S-Device Supervisor	30 <sub>hex</sub>	1	Exception Detail Alarm	13	STRUCT
Exception Detail Warning	S-Device Supervisor	30 <sub>hex</sub>	1	Exception Detail Warning	14	STRUCT
Pressure	S-Analog Sensor	31 <sub>hex</sub>	2	Indicated Pressure	6	INT
FP-Pressure	S-Analog Sensor	31 <sub>hex</sub>	2	Indicated Pressure	6	REAL
Temperature	S-Analog Sensor	31 <sub>hex</sub>	3	Indicated Temperature	6	INT
FP-Temperature	S-Analog Sensor	31 <sub>hex</sub>	3	Indicated Temperature	6	REAL

**6-42.9 Object Limitations and Specific Definitions**

This section describes limitations and specific definitions applicable to the listed objects of this device profile when these objects are applied in this type of device.

**6-42.9.1 S-Device Supervisor Object Instance**

The following limitations apply:

**Table 6-42.9 Device Type Attribute Limitations**

Attribute	Limitation
Device Type	Supported Values: “EMFC” = Enhanced Mass Flow Controller device “EMFM” = Enhanced Mass Flow Meter device

**Specific Definitions**

The following table specifies the data attribute bit mapping for the Device Exception Detail bytes for this EMFC device. For more descriptive information, see the definition of the S-Device Supervisor Object Class. Noted, for each entry, is the Object from which the Status byte/bit is mapped. See the object specification for the detailed bit mapping.

**Enhanced Mass Flow Controller, Type: 27<sub>hex</sub>**

Any Exception Bit not supported must default to 0. Note that this profile allows for only one byte of manufacturer specific exception detail.

Note that the status bits for the Pressure (Instance 2 of the S-Analog Sensor object) and the Temperature (Instance 3 of the S-Analog Sensor object) shall use the same bit mapping as the Flow instance (Instance 1 of the S-Analog Sensor object).

**Table 6-42.10 Device Exception Detail Mapping**

Data Component	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
MFC Device Exception Detail Size	0	0	0	0	0	0	1	0
MFC Device Exception Detail Byte 0	Reserved 0	Reserved 0	Valve High S-Analog Actuator	Valve Low S-Analog Actuator	Flow Control S-Single Stage Controller	Flow High S-Analog Sensor	Flow Low S-Analog Sensor	Not Reading Valid <sup>1</sup> S-Analog Sensor Flow
MFC Device Exception Detail Byte 1	Reserved 0	Reserved 0	Not Reading Valid S-Analog Sensor temperature	Not Reading Valid S-Analog Sensor pressure	Gas Temp High S-Analog Sensor Instance 3	Gas Temp Low S-Analog Sensor Instance 3	Pressure High S-Analog Sensor Instance 2	Pressure Low S-Analog Sensor Instance 2
Manufacturer Exception Detail Size	0	0	0	0	0	0	0	1
Manufacturer Exception Detail	0	0	0	0	0	0	0	0

1 Only used in the Warning Exception Detail, this bit is always = 0 in the Alarm Exception Detail. Refer to the S-Analog Sensor instance.

**6-42.9.2 S-Analog Sensor Object**

The following limitations apply:

**Table 6-42.11 S-Analog Sensor Object Attribute Limitations for Instance 1, (flow)**

Attribute	Limitation	Requirement	Default
Data Type	Supported Values = {INT; REAL}	Supported Values = {INT; REAL}	INT
Data Units	Supported Values = {Counts & Units of the Flow Group} (see Appendix D)	Supported Values = {Counts; sccm}	Counts
Offset-A Data Type	Supported Values = {INT; REAL}	Supported Values = {INT; REAL}	INT
Gain Data Type	Supported Values = {INT; REAL}	Supported Values = {INT; REAL}	INT

**Enhanced Mass Flow Controller, Type: 27<sub>hex</sub>****6-42.9.3 S-Analog Sensor Object**

The following limitations apply:

**Table 6-42.12 S-Analog Sensor Object Attribute Limitations for Instance 2, (pressure)**

Attribute	Limitation	Requirement	Default
Data Type	Supported Values = {INT; REAL}	Supported Values = {INT; REAL}	INT
Data Units	Supported Values = {Counts & Units of the Pressure Group} (see Appendix D)	Supported Values = {Counts; psi}	Counts
Offset-A Data Type	Supported Values = {INT; REAL}	Supported Values = {INT; REAL}	INT
Gain Data Type	Supported Values = {INT; REAL}	Supported Values = {INT; REAL}	INT

**6-42.9.4 S-Analog Actuator Object**

The following limitations apply:

**Table 6-42.13 S-Analog Actuator Object Attribute Limitations**

Attribute	Limitation	Requirement	Default
Data Type	Supported Values = {INT; REAL}	Supported Values = {INT; REAL}	INT
Data Units	Supported Values = {Counts, %, Voltage, Current & Units of the Flow Group} (see Appendix D)	Supported Values = {Counts; % }	Counts
Gain Data Type	Supported Values = {INT; REAL}	Supported Values = {INT; REAL}	INT

**6-42.9.5 S-Single Stage Controller Object**

The following limitations apply:

**Table 6-42.14 S-Single Stage Controller Object Attribute Limitations**

Attribute	Limitation	Requirement	Default
Data Type	Supported Values = {INT; REAL}	Supported Values = {INT; REAL}	INT
Data Units	Supported Values = {Counts, %, Voltage, Current & Units of the Flow Group} (see Appendix D)	Supported Values = {Counts; % }	Counts
Process Variable	Not accessible over the network. The <i>Process Variable</i> input to this object instance is the value of the S-Analog Sensor object instance <i>Value</i> attribute.	N.A.	N.A.
CV Data Type	Not supported	N.A.	N.A.
Control Variable	Not accessible over the network. The <i>Control Variable</i> output from this object is the value of the S-Analog Actuator object instance <i>Value</i> Attribute.	N.A.	N.A.

Note - % refers to a vendor specific value that equates to 0 to 100% of full scale

**6-42.10 Defining Device Configuration**

Public access to the S-Device Supervisor, S-Analog Sensor, S-Analog Actuator, S-Single Stage Controller, and S-Gas Calibration Objects by the Message Router must be supported for configuration of this device type.

## **Volume 1: Common Industrial Protocol (CIP<sup>TM</sup>) Specification**

### **Chapter 7: Electronic Data Sheets**

---

## Contents

7-1	Introduction.....	4
7-1.1	Terms and Definitions Used in this Chapter .....	4
7-2	Configuration Options .....	5
7-2.1	Configuration Data Sources and Methods .....	5
7-2.1.1	Configuration Support Using Printed Data Sheets.....	5
7-2.1.2	Configuration Support Using an Electronic Data Sheet.....	6
7-2.1.3	Configuration Support Using Parameter Objects and Parameter Object Stubs .....	7
7-2.1.3.1	Using Full Parameter Objects .....	7
7-2.1.3.2	Using Parameter Object Stubs .....	8
7-2.1.4	Configuration Using an EDS and Parameter Object Stubs .....	8
7-2.1.5	Configuration Using a Configuration Assembly.....	9
7-2.1.6	Configuration Using a Device Type Manager (DTM).....	10
7-2.1.6.1	Developing a Device Type Manager (DTM).....	10
7-3	Electronic Data Sheet Definition .....	11
7-3.1	The Electronic Data Sheet.....	11
7-3.2	The Product Data Sheet Model .....	12
7-3.3	Using an EDS with Configuration Tools .....	15
7-3.3.1	EDS Interpreter Functions .....	15
7-3.3.2	EDS File Organization .....	15
7-3.3.2.1	EDS Content .....	16
7-3.4	File Naming Requirements .....	18
7-3.5	EDS Data Encoding Requirements .....	19
7-3.5.1	ASCII Character File Convention .....	19
7-3.5.2	Character String Convention - (STRING) .....	19
7-3.5.3	String Concatenation.....	20
7-3.5.4	String Escape Sequences.....	20
7-3.5.5	ASCII Unsigned Integer Convention - (USINT, UINT, UDINT, ULINT) .....	21
7-3.5.6	ASCII Signed Integer Convention - (SINT, INT, DINT, LINT) .....	21
7-3.5.7	ASCII Word Convention - (BYTE, WORD, DWORD, LWORD) .....	22
7-3.5.8	CIP Path .....	22
7-3.5.9	STRINGI.....	23
7-3.5.10	EDS Comments.....	24
7-3.5.11	EDS Date .....	24
7-3.5.12	EDS Time Of Day .....	24
7-3.5.13	ASCII Floating Point Convention (REAL, LREAL) .....	25
7-3.5.14	EDS_URL Uniform Resource Locator .....	26
7-3.6	Basic EDS File Requirements.....	27
7-3.6.1	Common Object Class Information.....	27
7-3.6.1.1	Object Class Sections.....	27
7-3.6.1.2	Common Object Class Entry Keywords .....	30
7-3.6.1.3	Parameter Class Section.....	33
7-3.6.1.4	Connection Configuration Class Section .....	35
7-3.6.2	File Description Section.....	39
7-3.6.3	Device Description Section.....	42
7-3.6.4	Device Classification Section .....	44
7-3.6.5	Parameters Section .....	45
7-3.6.6	Parameter Groups Section.....	47
7-3.6.7	Assembly Section.....	48
7-3.6.7.1	Revision Entry Keyword .....	48
7-3.6.7.2	Assem type Entry Keywords .....	48
7-3.6.7.3	Variant and VariantExa Entry Keywords .....	54
7-3.6.7.4	Examples of Adapter/Rack Assem Data Formatting .....	58

7-3.6.8	Complete Electronic Data Sheet Example .....	63
7-3.6.9	Connection Manager Section .....	64
7-3.6.9.1	Trigger and transport mask .....	65
7-3.6.9.2	Connection parameters .....	65
7-3.6.9.3	O->T RPI .....	67
7-3.6.9.4	O->T size .....	67
7-3.6.9.5	O->T format .....	67
7-3.6.9.6	T->O RPI .....	67
7-3.6.9.7	T->O size .....	67
7-3.6.9.8	T->O format .....	68
7-3.6.9.9	Configuration .....	68
7-3.6.9.10	Connection Name String .....	68
7-3.6.9.11	Help string .....	68
7-3.6.9.12	Path .....	68
7-3.6.10	Port Section .....	69
7-3.6.11	Capacity Section .....	71
7-3.6.11.1	Traffic Spec .....	71
7-3.6.11.2	Generating TSpec Values .....	72
7-3.6.11.3	Connection Overhead .....	73
7-3.6.11.4	Determining ConnOverhead Values .....	74
7-3.6.11.5	Capacity Assumptions .....	75
7-3.6.11.6	Calculating Implicit Communications Usage .....	75
7-3.6.11.7	Connection Capacity Keywords .....	76
7-3.6.11.8	Considerations for Multiport Devices .....	77
7-3.6.12	Event Enumeration Section .....	78
7-3.6.13	Symbolic Translation Section .....	81
7-3.6.14	Internationalization Section .....	83
7-3.6.14.1	Descriptive Text Entry Keyword .....	84
7-3.6.14.2	Device Type String Entry Keyword .....	84
7-3.6.14.3	Product Name Entry Keyword .....	84
7-3.6.14.4	Parameter Entry Keywords .....	85
7-3.6.14.5	Enumeration Entry Keyword .....	85
7-3.6.14.6	Group Entry Keyword .....	86
7-3.6.14.7	Assembly Entry Keywords .....	86
7-3.6.14.8	Variant Entry Keywords .....	86
7-3.6.14.9	Connection Entry Keyword .....	87
7-3.6.14.10	Port Entry Keyword .....	87
7-3.7	Modular EDS File Requirements .....	88
7-3.7.1	Modular Section .....	88
7-3.7.2	Modular Additions to Basic EDS Sections .....	90
7-3.7.2.1	CIP Path Addition for Modular EDS Files .....	90
7-3.7.2.2	Additions to the Modular Section .....	90
7-3.7.2.3	Additions to the Parameter Section .....	91
7-3.7.2.4	Additions to the Assembly Section .....	92
7-3.7.2.5	Additions to the Connection Manager Section .....	95

## **7-1 Introduction**

This chapter describes:

- standardized methods for device configuration
- the structure of an Electronic Data Sheet (EDS)

The information provides reference material for:

- product developers
- configuration tool designers

### **7-1.1 Terms and Definitions Used in this Chapter**

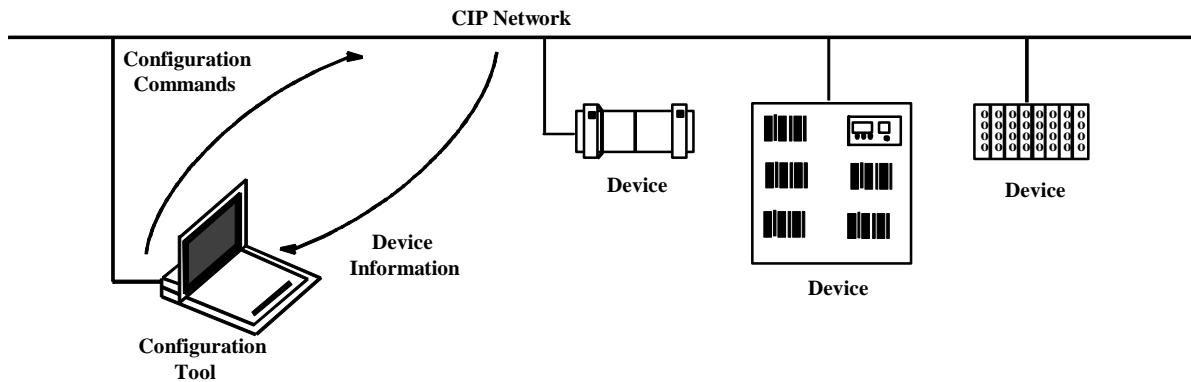
**Table 7-1.1 Terms and Definitions**

<b>Term</b>	<b>Meaning</b>
decoded format	The decoded attribute data values in CIP message format
EDS	The abbreviation for Electronic Data Sheet, a file on disk that contains configuration data for specific device types
encoded	The encoded attribute data values in the Electronic Data Sheet Format
CIP path	The address of an object attribute in CIP Class-Instance-Attribute format
full parameter object	A parameter object instance within a device that contains the data value, prompt strings, data conversion factors, and other information associated with the parameter
stub parameter object	A shorthand form of a parameter object instance that stores only fields 1-6
Rack	Typically a multi-slot chassis that contains a variety of modules
Adapter	A device connected directly to a CIP network, which can be placed in one or more slots of a rack
Module	A device that can be placed in one or more slots of a rack
Adapter Rack Connection	A connection to an adapter whose O->T and T->O application data includes data produced/consumed by one or more modules

## **7-2 Configuration Options**

The CIP standard allows for remote device configuration across the network and for embedding configuration parameters in devices. Using these features, you can select and modify device configuration settings for use in a particular application. The CIP interface allows access to a device's configuration settings.

**Figure 7-2.1 Illustration of Remote Device Configuration**



Devices containing configuration settings accessible only through the CIP communications interface require a configuration tool to change these settings. Devices configured only through the use of external switches, jumper blocks, thumbwheels, or other proprietary interfaces do not require a configuration client (tool) to modify the device's configuration settings. However, the device designer must provide a means for a tool to access and determine the state of hardware configuration switches.

### **7-2.1 Configuration Data Sources and Methods**

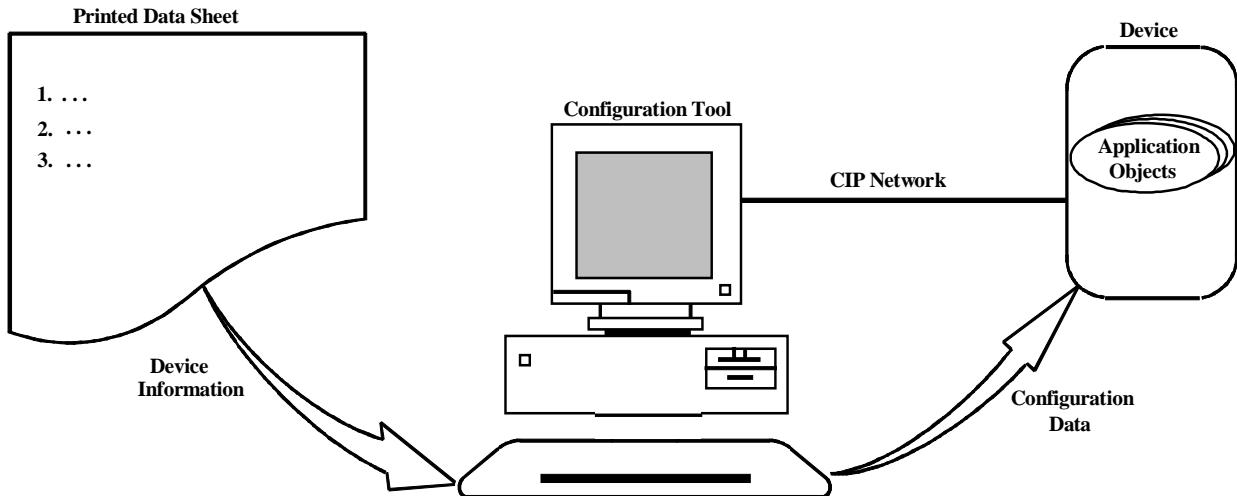
This section describes possible methods for storing and accessing a device's configuration data

- A printed data sheet
- An Electronic Data Sheet (EDS)
- Parameter Objects and Parameter Object Stubs
- A combination of an EDS and Parameter Object Stubs
- A Configuration Assembly and any of the above methods
- A Device Type Manager (DTM) loaded into a Field Device Tool software package

#### **7-2.1.1 Configuration Support Using Printed Data Sheets**

When using configuration information collected on a printed data sheet, configuration tools can only provide prompts for service, class, instance, and attribute data and relay this data to a device. A configuration tool of this type does not determine the context, content, or format of the data. The figure below shows the relationship between configuration information and the tool.

**Figure 7-2.2 Illustration of Printed Data Sheet Support**



### 7-2.1.2 Configuration Support Using an Electronic Data Sheet

You can provide configuration support for your device by using a specially formatted ASCII file, referred to as the *Electronic Data Sheet* (EDS). An EDS provides information about the device configuration data's:

- context
- content
- format

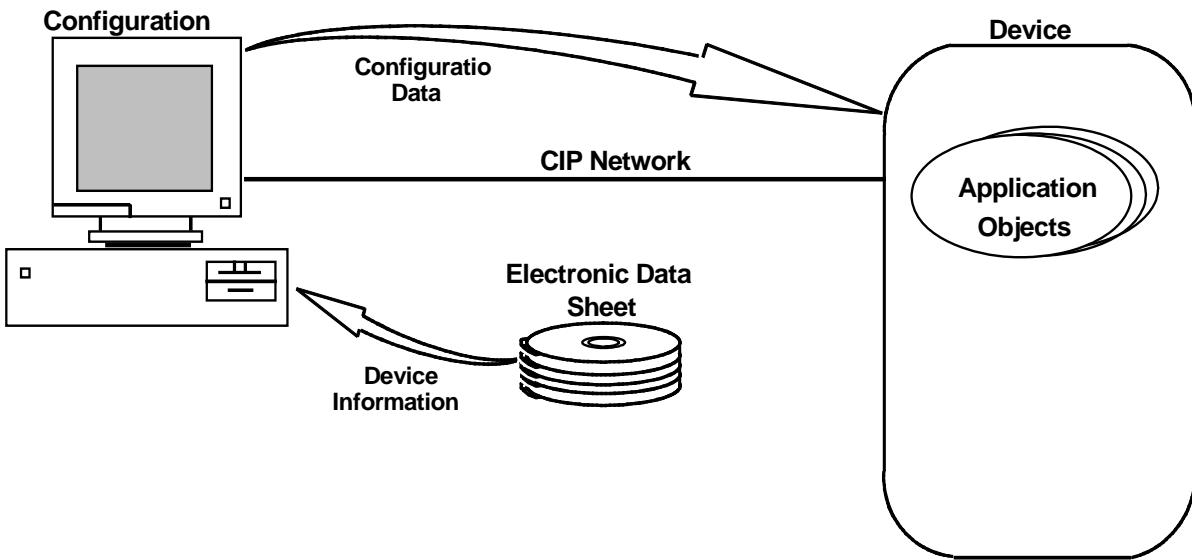
The information in an EDS allows configuration tools to provide informative screens that guide a user through the steps necessary to configure a device. Section 7-3 describes electronic data sheets.

An EDS provides all of the information necessary to access and alter the configurable parameters of a device. This information matches the information provided by instances of the Parameter Object Class. The CIP Object Library describes the Parameter Object Class in detail.

Manufacturers who choose not to supply an EDS on computer-readable media can provide a printed listing of their EDS so that the end user may create the computer-readable EDS using a text editor.

The following figure shows device configuration by a configuration tool that supports an EDS. The application objects in the device represent the destination addresses for configuration data. These addresses are encoded in the EDS.

Figure 7-2.3 Illustration of Using Electronic Data Sheets for Configuration



### 7-2.1.3 Configuration Support Using Parameter Objects and Parameter Object Stubs

The device's public *Parameter Object*, which is an optional data structure in a device, provides a third method for accessing the configuration data of a device. When a device uses parameter objects, it requires one instance of the Parameter Object class for each supported configuration parameter. Each instance is linked to a configurable parameter that can be an attribute of one of the device's other objects. Changing the Parameter Value attribute of a Parameter Object causes a corresponding change in the attribute value, as indicated by the Link Path attribute.

A *Full Parameter Object* contains all of the information necessary for device configuration. The CIP Object Library, contains a complete definition of the Parameter Object. A partially defined Parameter Object, called a *Parameter Object Stub*, contains the portion of configuration information needed to perform configuration that excludes user prompts, limit testing, and descriptive text that guides a user through the configuration.

#### 7-2.1.3.1 Using Full Parameter Objects

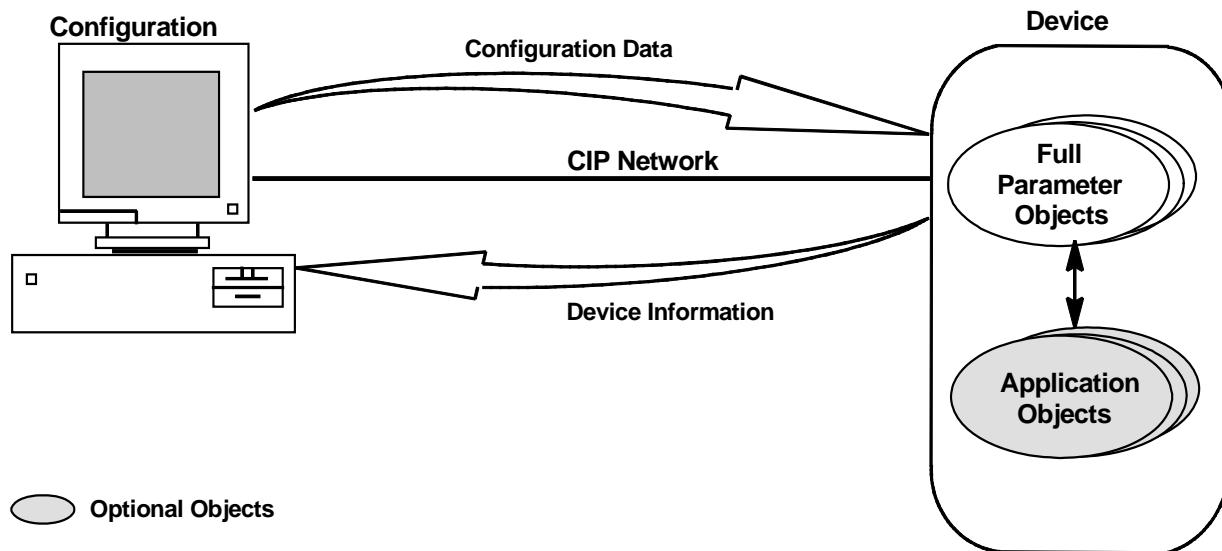
Parameter Objects embed all of the needed configuration information within the device. Parameter Objects provide:

- a known public interface to a device's configuration data values
- descriptive text
- data limits, default, minimum, and maximum values

**Important:** When a device contains full Parameter Objects, a configuration tool may derive all of the needed configuration information directly from the device.

The following figure shows the configuration of a device by a configuration tool that supports full Parameter Objects.

**Figure 7-2.4 Illustration of Using Parameter Objects To Configure a Device**



#### 7-2.1.3.2 Using Parameter Object Stubs

Parameter Object stubs provide an established address to a device's configuration data values without requiring specification of descriptive text, data limits, and other parameter properties. When a device contains Parameter Object stubs, a configuration tool may obtain additional configuration information from an EDS or provide only a minimal interface to a parameter for modification.

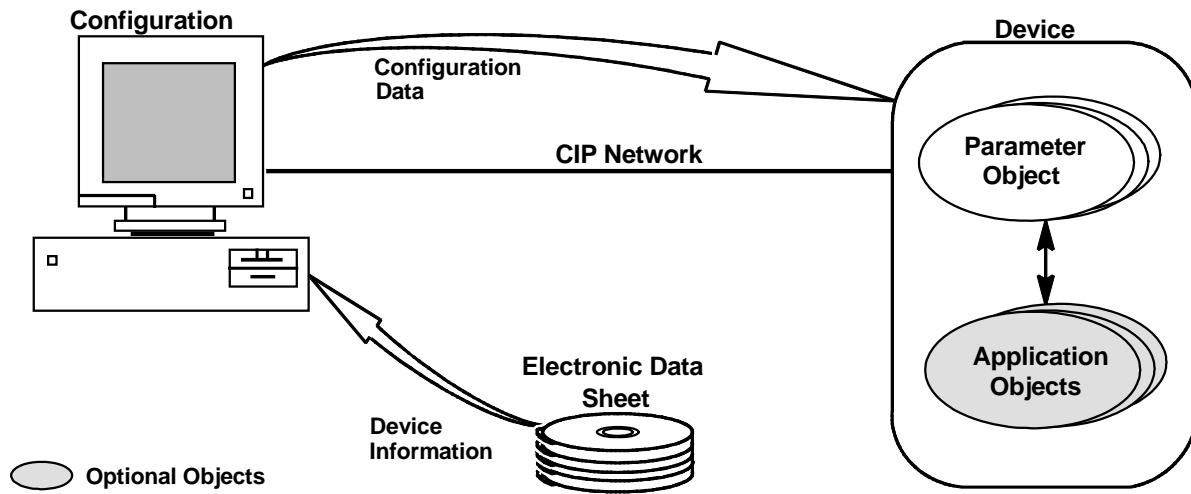
#### 7-2.1.4 Configuration Using an EDS and Parameter Object Stubs

A configuration tool may obtain information from partial Parameter Objects or Parameter Object stubs embedded in a device that provides a companion EDS. The EDS supplies additional parameter information needed by a configuration tool. The Parameter Object stubs can provide a known public interface to a device's parameter data values, while the EDS supplies descriptive text, data limits, and other parameter properties such as the:

- Data type and size for data validation
- Default data selections
- Descriptive user prompts
- Descriptive help text
- Descriptive parameter names

The figure below shows the configuration of a device by a configuration tool that supports Parameter Object stubs with electronic data sheets.

Figure 7-2.5 Illustration of EDS and Parameter Object Stubs to Configure a Device



### 7-2.1.5 Configuration Using a Configuration Assembly

An instance of the Configuration Assembly allows bulk upload and download of configuration data. If you use this method for device configuration, you shall document:

- The format of the configuration data blocks
- The address mapping for each configurable attribute

When specifying the data attribute of the Configuration Assembly, you shall:

- List the data components in the order given by the attribute block
- List data components larger than one byte starting with the low order byte
- List data components smaller than one byte justified to the right within a byte, starting with bit 0

Show the format of the data blocks in a table like the following:

Table 7-2.1 Example of Configuration Assembly Data Attribute

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Operate Mode
	o o o							
n	Input Range							

In addition to specifying the device's Configuration Assembly's data format, you shall map the individual configuration data components to their associated objects. Do this by specifying the Class, Instance, and Attribute IDs for each data component. This is essentially the same as specifying the Member\_List attribute of the Assembly Object.

The table below shows an example mapping of Configuration Assembly data attribute components:

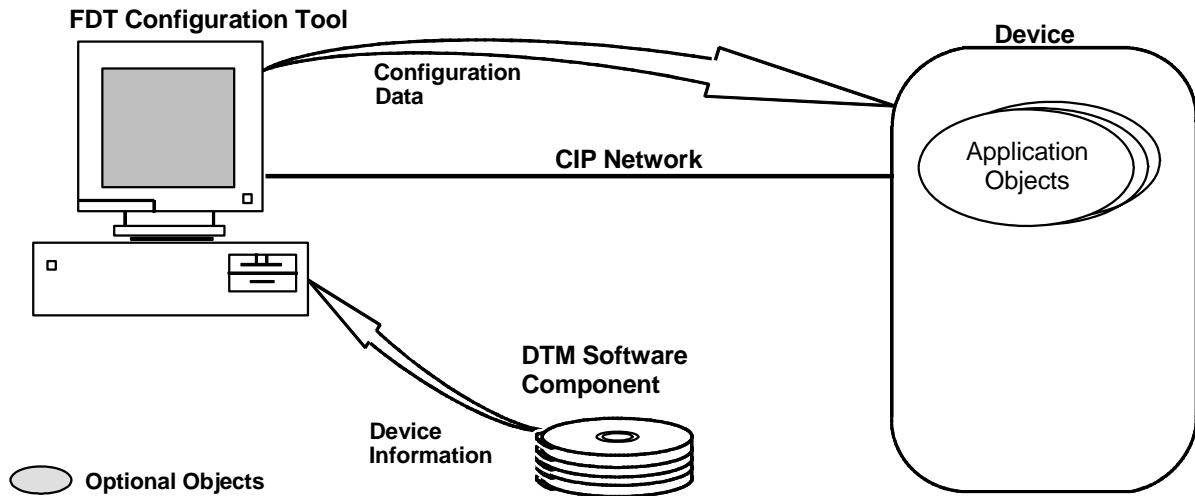
**Table 7-2.2 Example of Configuration Assembly Data Attribute Components**

Data Component Name	Class		Instance Number	Attribute		Data Type
	Name	Number		Name	Number	
Output Range	Analog Output	0B hex	1	Output Range	7	UINT
		o				
		o				
		o				
Input Range	Analog Input	0A hex	1	Input Range	7	BYTE

### 7-2.1.6 Configuration Using a Device Type Manager (DTM)

A device can be configured by use of a Device Type Manager (DTM) software component. The DTM is loaded into a configuration software package that supports the Field Device Tool (FDT) interfaces to the DTM. This is demonstrated in below.

**Figure 7-2.6 Configuration Using a DTM Software Component**



Like an EDS file, the DTM contains all of the device specific information for configuring the device. However, while the EDS file is a structured text file describing the data objects, the DTM is a software component that adheres to the FDT interfaces. This allows a greater level of flexibility to add device specific functionality into the configuration interface than can be achieved in an EDS file.

#### 7-2.1.6.1 Developing a Device Type Manager (DTM)

The controlling specifications for the development of a DTM are in the FDT specification and the supporting annex for CIP networks. See [www.fdtgroup.org](http://www.fdtgroup.org) for details.

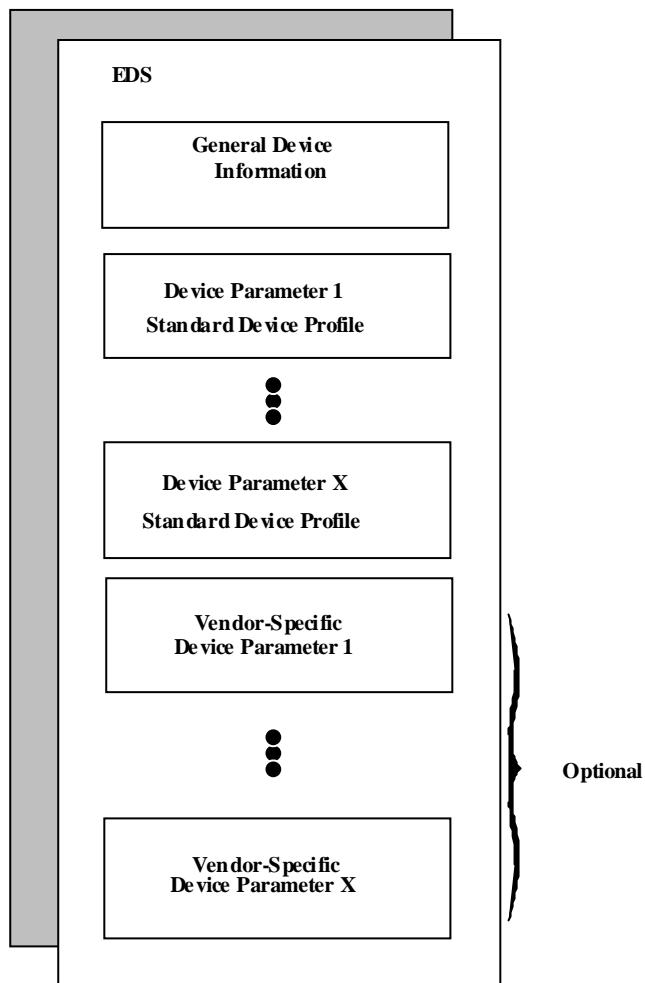
## 7-3 Electronic Data Sheet Definition

This section contains the formal specification of the electronic data sheet (EDS). The EDS allows a configuration tool to automate the device configuration process. The EDS specification provides an open standard for device configuration and compatibility among all CIP products.

### 7-3.1 The Electronic Data Sheet

An EDS may contain vendor-specific information in addition to the required device parameter information defined by this specification. The figure below shows a general block diagram of a sample EDS.

**Figure 7-3.1 General Block Diagram of an EDS File**



### 7-3.2 The Product Data Sheet Model

Electronic Data Sheets follow a product data sheet metaphor, modified to accommodate CIP requirements. Typically, a product data sheet provides a user with information needed to determine the product's features and the range of user-assigned values that may be selected for these features.

**Figure 7-3.2 Example of a Product Data Sheet**

Product Name:	Smart Widget I1 Series Z99 Dual Mode	<= Name of the product
Vendor:	Always Profitable Widget Works, Inc.	<= Who made it
Model:	Z99 DM	<= Catalog number
Revision:	2.1	<= Revision level
Serial Number:	00123456789A	<= Unique serial number

User Settings	Minimum	Maximum	Factory Default	Access Point
1. Gain	1	64	32	SW1.8
2. Scale Factor	1	8	1	SW4
3. Output Mode	RTZ	NRTZ	RTZ	Jumper Block 5
4. Output Frequency	1 Hz	100 Hz	30 Hz	SW2
5. Status Register	N/A	N/A	N/A	LED2

The data sheet conveys information from the product manufacturer to the product user. The product user interprets the manufacturer's data sheet, decides which settings must be set to non-default values, and performs the necessary actions to get the information from the data sheet into the device.

A configuration tool that uses an EDS and parameter object stubs follows this same sequence, using an electronic form of the data sheet.

To perform the actual configuration, a configuration tool uses CIP messaging to perform the changes in the device. Currently, the textual information in an EDS must be ASCII-representable characters. Future revisions of this specification will support non-ASCII character representations such as UNICODE.

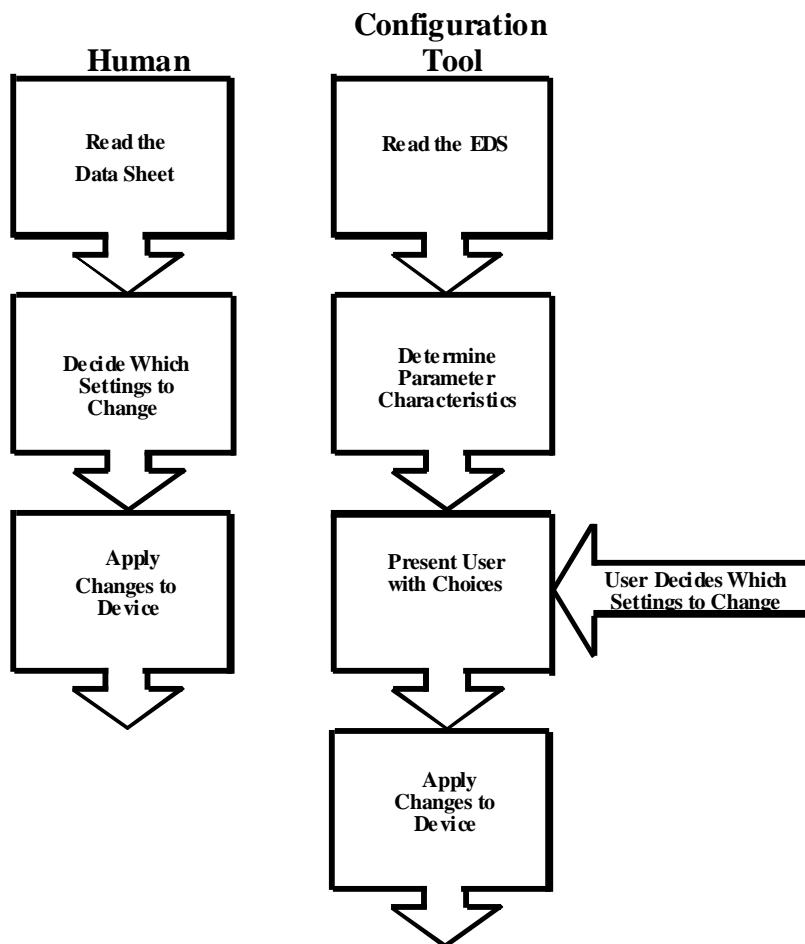
The EDS serves two main purposes by providing information needed to:

- Describe each device parameter, including its legal and default values
- Provide the user with a selection of choices for each configurable parameter in a device

The figure below compares the interpretation of a printed data sheet by a human with the interpretation of an EDS by a configuration tool. At the minimum, a CIP configuration tool must:

- Load the EDS into the configuration tool's memory
- Interpret the EDS contents to determine the characteristics of each parameter
- Present to the user a list of choices or data entry fields for each device parameter
- Load the user's parameter choices to the correct parameter addresses in the device

**Figure 7-3.3 Illustration of Using Printed Data Sheets vs EDS**



Product developers may add optional EDS-related functions to a configuration tool. The specified EDS requirements provide a consistent and compatible approach for performing configuration in the CIP environment. This specification enforces only the requirements critical to achieving:

- Consistency among vendors for the benefit of device makers
- Compatibility among vendors for the benefit of tool makers

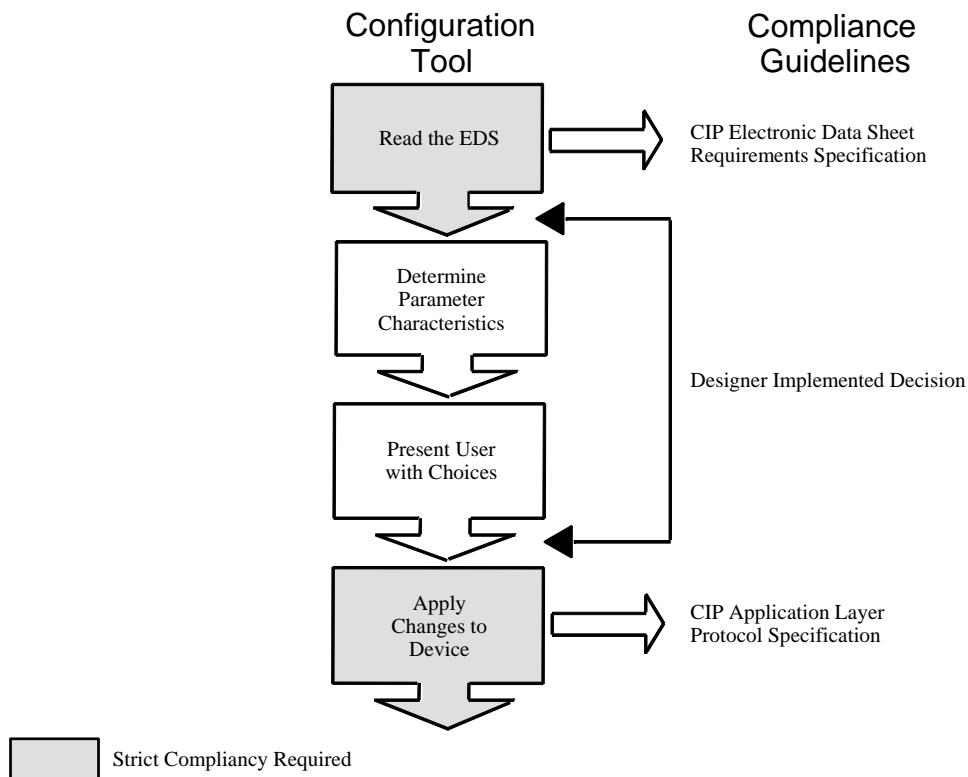
All EDS developers shall implement EDSs compliant with these requirements. Product developers determine all other implementation details. Every EDS interpreter designed for CIP products shall be capable of:

- Reading and interpreting any standard EDS
- Presenting information and choices to device users
- Building the messages necessary to configure the associated CIP product

The specification, in CIP Communication Model and Protocol, imposes strict requirements and protocols for transmitting messages over the CIP network. The figure below shows the compliancy required for each configuration process. The remainder of this chapter describes the requirements which shall be satisfied for all configuration tools that use an EDS. These requirements describe:

- EDS encoding
- EDS command language
- EDS syntax

**Figure 7-3.4 Compliancy Requirements for the Configuration Process**



### **7-3.3 Using an EDS with Configuration Tools**

CIP configuration tools:

- Extract user prompt information from a standard EDS
- Present this information to the user in a human-readable form

The tool developer determines the:

- user interface format
- operator interaction

#### **7-3.3.1 EDS Interpreter Functions**

The interpreter shall:

- Collect parameter selections required by an EDS
- Build the CIP messages necessary to configure a device
- Contain the object addresses for each device parameter requiring configuration

#### **7-3.3.2 EDS File Organization**

This section contains the file encoding requirements for EDSs. The EDS encoding requirements define the standard file encoding format to use for CIP products without regard to the configuration tool host platform or file system.

The term “file” as used in this chapter refers to any recognized file format associated with a configuration tool’s file system without regard to the file storage media. An EDS contains an ASCII representation of a device’s Parameter Objects and some additional information required to support object addressing.

### 7-3.3.2.1 EDS Content

A single file must contain the entire EDS. The following table summarizes the structure of the sections, legal section delimiters, and the order of sections in an EDS.

**Table 7-3.1 Sections of an EDS File**

Section Name	Legal Delimiter	Placement	Required/Optional
File Description	[File]	1	Required
Device Description	[Device]	2	Required
I/O Characteristics	[IO_Info]	See Note 1	DeviceNet specific. See Vol 3, DeviceNet Adaptation of CI for details
Enumerated Parameters	[EnumPar]	See Note 1	DeviceNet specific. See Vol 3, DeviceNet Adaptation of CIP for details
Parameter Class	[ParamClass]	See Note 1	Optional
Parameters	[Params]	See Note 1	Optional
Parameter Groups	[Groups]	See Note 1	Optional
Assembly	[Assembly]	See Note 1	Optional
Device Classification	[Device Classification]	See Note 1	Optional
Connection Characteristics	[Connection Manager]	See Note 1	Optional
Port	[Port]	See Note 1	Optional
Modular	[Modular]	See Note 1	Optional
Variant_IO_Info	[Variant_IO_Info]	See Note 1	DeviceNet specific. See Vol 3, DeviceNet Adaptation of CIP for details
Capacity	[Capacity]	See Note 1	Optional
Connection Configuration Class	[Connection Configuration]	See Note 1	Conditional (See Note 2)
Event Enumeration	[EventEnum]	See Note 1	Optional
Symbolic Translation	[SymbolicTranslation]	See Note 1	Optional
Object class sections	See Chapter 7-3.6.1.1		
Internationalization	[Internationalization]	See Note 1	Optional

Note 1 - Placement of optional groups only needs to follow the required groups.

Note 2 – Required if the device supports the Connection Configuration object, not allowed if the device does not support the Connection Configuration object.

Observe these rules when defining an EDS:

- EDS White Space - The EDS interpreter shall treat certain characters as white space characters. These characters, read by the interpreter but not encoded as human-readable characters, designate the presence of blank space in a file. White space characters are:
  - New line
  - Carriage Return
  - Linefeed
  - Tabs, vertical and horizontal
  - Form Feed
  - End of File marker

- Comments
- Keyword characters – All keywords within an EDS file must be composed of ASCII characters from the following list:
  - Upper case letters A through Z
  - Lower case letters a through z
  - Numerals 0 through 9
  - The special character underscore "\_"
- The space character (The space may only be used in a section keyword. The space may only appear internal to a section name, and multiple sequential spaces are invalid)
- Sections - The EDS file must be partitioned into required and optional sections.
- Section delimiters - Each section in the EDS must be properly delimited by a *section keyword*.
- Section Keywords – Each section keyword is defined to be the text between the beginning of section keyword delimiter "[" and the terminating delimiter "]". The characters valid for use in section keywords are shown above. Section keywords shall not be case sensitive. There are two types of section keywords, public and vendor specific.
- Section order - Each required section must be placed in the required order.
- Entry - Each section in the EDS contains one or more *entries* beginning with an *entry keyword* followed by an equal sign. The entry keyword meaning depends on the context of the section. A semicolon indicates the end of the entry, and entries may span multiple lines.
- Entry Keywords – An entry keyword consists of a unique sequence of keyword characters. Entry keywords shall not be case sensitive. There are two types of entry keywords, public and vendor specific.
- Public Keyword – A Public Keyword shall always be defined by ODVA and shall be listed within Chapter 7 of Volume 1, Common Industrial Protocol Specification. A public keyword shall never begin with any numeric digit.
- Entry fields - Each entry contains one or more fields. A comma delimiter separates each field.
  - A white space or nothing between commas (for example: keyword = x,,x;).
  - A white space or nothing between a comma and a semi-colon (for example: keyword = x,;).
  - A white space or nothing between an equal and comma (for example: keyword = ,x;).
  - A white space or nothing between an equal and semi-colon (for example: keyword = ;).Trailing optional fields can be omitted by ending the Entry with a semicolon. The meaning of the field(s) depends on the context of the section.
- Complex data fields – Certain entry fields must be specified with data that cannot be specified by a single value between the comma delimiters. The ability to further delimit an entry field is defined via the use of one or more sets of matching brace characters "{" and "}". The content between brace characters is considered a single item or entry. Content may be grouped in multiple braces.

- Vendor-Specific Keywords - Section and Entry Keywords may be vendor-specific. Vendor Specific Section Keywords shall begin with the Vendor ID of the company making the addition followed by an underscore (*VendorID\_VendorSpecificKeyword*). Vendor Specific Entry Keywords not in a vendor-specific section shall begin with the Vendor ID of the company making the addition followed by an underscore (*VendorID\_VendorSpecificKeyword*). Vendor Specific Entry Keywords in a vendor-specific section are always vendor-specific; they are not required to begin with the Vendor ID of the company making the addition. The VendorID shall be displayed in decimal and shall not contain leading zeroes. Each vendor is responsible for maintaining and documenting their vendor-specific keywords.

The following examples highlight the structure of the electronic data sheet (Note: The “\$” character denotes a comment.)

**Figure 7-3.5 Basic Structure of an EDS file**

```
[section name]
$ Comment - extends to end of line

Entry1=Field1, Field2, Field3;           $ Entire entry on one line

Entry2=Field1, Field2, Field3, Field4;   $ Entire entry on one line

Entry3=          $ Multiple line entry
    Field1,           $ Field1
    Field2,           $ Field2
    Field3;          $ Field3

Entry4=          $ Combination
    Field1, Field2,      $ Fields 1 and 2 on one line
    Field3,           $ Field3
    Field4;          $ Field4

65535_Entry= $ Vendor Specific entry for Vendor_ID 65535
    Field1, Field2;     $ with two fields

Entry5 = 1,           $ Field 1 specifies the value 1
    {1,2,3};           $ Field 2 specifies an array or
                        $ structure with three values

Entry6 = { 44, {22,33,11} };    $ Entry 6 specifies a single field.
                                $ The field contains two sets of data.
                                $ The first set is a single value 44.
                                $ The second set contains three values.
```

Note from the examples that an entry can extend over multiple lines as long as commas properly delimit the fields. The configuration tool ignores any whitespace characters, including comments, tabs, and spaces. Comments start at the comment delimiter (\$) and extend to the end of the line. All entries must terminate with a semicolon.

### **7-3.4 File Naming Requirements**

Currently no file naming conventions exist for disk-based EDS files, except for files in a DOS/Windows environment. The file should have the suffix “.EDS” appended to the file name.

## **7-3.5 EDS Data Encoding Requirements**

This section describes the data encoding requirements for the EDS file.

### **7-3.5.1 ASCII Character File Convention**

All data in the EDS must be encoded using 8-bit ASCII characters, where all references to “ASCII characters” mean an 8-bit ASCII character format (as defined by Tables 1 and 2, Row 00 of the IEC/ISO 10646-1). Characters that cannot be displayed on an ANSI terminal cannot be used in identifier names or in data representations.

### **7-3.5.2 Character String Convention - (STRING)**

All references to STRING in this chapter mean ASCII character strings of fixed length without null terminators. Double quotes enclose all strings.

Characters between double quotation marks (“) are interpreted as string data. There are two forms of string data conversions. Characters contained between double quotation marks are converted into 8-bit characters. Characters contained between double quotation marks that are preceded by a capital L are converted into UNICODE (16-bit) characters. The following are examples:

- “This results in a string composed by 8-bit characters”.
- L”A string of UNICODE characters, including the Greek character Pi \u03C0”

Note: the text \u03C0 specifies a single 16-bit character whose value is 03C0. In the UNICODE character set, this is Table 9, Row 3, Basic Greek – the character for lower case "Pi". Descriptions of character escape sequences are described in subsequent section 7-3.5.4.

#### **Handling Insufficient Characters in a String Field**

An EDS interpreter uses right-justification of characters in a field and fills any unspecified characters with leading blanks (ASCII 0x20) for the remaining length of the string. For example, if a parameter has a maximum string length of 8 and receives the string

”123AB”,

the interpreter decodes the string as:

”~~~123AB”,

where the tilde characters (~) represent spaces.

#### **Handling Excess Characters in a String Field**

If a given string field contains too many characters, the EDS interpreter truncates characters from left to right. For example, if a parameter has a maximum string length of 8 and receives the string

”I23ABCDEFG”,

the EDS interpreter truncates the string to

”I23ABCDE”.

### **7-3.5.3 String Concatenation**

Multiple strings with no intervening commas are concatenated. For example, the line

”ABC” ”123” ”XYZ”

is interpreted as:

”ABC123XYZ”

The strings may also be on separate lines. For example, the following lines

”ABC” \$ this is a comment

”123”

”XYZ”

are also interpreted as:

”ABC123XYZ”

For a UNICODE string (long string), only the first double quotation mark shall be preceded by a capital L. Example: L”ABC” ”123” ”XYZ” is the same as L”ABC123XYZ”.

### **7-3.5.4 String Escape Sequences**

The EDS interpreter shall recognize all listed sequences. Interpretation is application specific.  
The escape sequences are:

\\" translates to a \

\n translates to a newline

\t translates to a tab

\v translates to a vertical tab

\b translates to a backspace

\r translates to a carriage return

\f translates to a form feed

\a translates to the BELL character (0x07)

\? translates to a ?

\0 translates to a null

\” translates to a ”

\' translates to ’

\xnn translates to a single byte containing the value of “nn” as expressed in hexadecimal

\unnnn translates to a pair of bytes containing the value of “nnnn” as expressed in hexadecimal. This form of string escape is only valid where the resultant string data is 16-bits in length, e.g. the L” form of string specification.

If a sequence not listed above is encountered, the interpreting device shall reject the entire string and indicate an error. EDS files shall only contain escape sequences defined above.

### **7-3.5.5 ASCII Unsigned Integer Convention - (USINT, UINT, UDINT, ULINT)**

The unsigned integer data types represent positive integer values. Unsigned integer data may be entered in decimal or hexadecimal notation with no white space or commas between characters. If hexadecimal notation represents the unsigned integer characters, the two character sequence 0x, with no white space, shall precede the unsigned integer characters.

The range of legal **USINT** data is:

Decimal Notation:	0	to	255
Hexadecimal Notation:	0x0	to	0xFF

The range of legal **UINT** data is:

Decimal Notation:	0	to	65535
Hexadecimal Notation:	0x0	to	0xFFFF

The range of legal **UDINT** data is:

Decimal Notation:	0	to	4294967295
Hexadecimal Notation:	0x0	to	0xFFFFFFFF

The range of legal **ULINT** data is:

Decimal Notation:	0	to	18446744073709551615
Hexadecimal Notation:	0x0	to	0xFFFFFFFFFFFFFF

### **7-3.5.6 ASCII Signed Integer Convention - (SINT, INT, DINT, LINT)**

The SINT, INT, DINT and LINT data types represent signed integer data values. Signed integer data may be entered in decimal or hexadecimal notation with no white space or commas between characters. If hexadecimal notation represents the signed integer characters, the two character sequence 0x, with no white space, shall precede the integer value characters.

The range of legal **SINT** data is:

Decimal Notation:	-128	to	127
Hexadecimal Notation:	0x80	to	0x7F

The range of legal **INT** data is:

Decimal Notation:	-32768	to	32767
Hexadecimal Notation:	0x8000	to	0x7FFF

The range of legal **DINT** data is:

Decimal Notation: -2147483648 to 2147483647

Hexadecimal Notation: 0x80000000 to 0xFFFFFFFF

The range of legal LINT data is:

Decimal Notation: -9223372036854775808 to 9223372036854775807

Hexadecimal Notation: 0x8000000000000000 to 0x7FFFFFFFFFFFFFFF

### **7-3.5.7 ASCII Word Convention - (BYTE, WORD, DWORD, LWORD)**

The BYTE, WORD, DWORD and LWORD data types represent bit-addressable values. These values are considered discrete bit position values and are not intended to represent either signed or unsigned integer values. However, these values, for convenience, may be entered in decimal, hexadecimal or binary notation with no white space or commas between characters. If hexadecimal notation represents value characters, the two character sequence 0x, with no white space, shall precede the value characters.

The range of legal **BYTE** data is:

Decimal Notation:                  0            to        255

Hexadecimal Notation: 0x0 to 0xFF

Binary Notation: 0b00000000 to 0b11111111

The range of legal **WORD** data is:

Decimal Notation:                  0            to        65535

Hexadecimal Notation: 0x0 to 0xFFFF

Binary Notation: 0b0000000000000000 to 0b1111111111111111

The range of legal **DWORD** data is:

Decimal Notation: 0 to 4294967295

Hexadecimal Notation: 0x0 to 0xFFFFFFFF

Binary Notation: 0b00000000000000000000000000000000 to  
0b111111111111111111111111111111111111111

The range of legal **LWORD** data is:

Decimal Notation: 0 to 18446744073709551615

Hexadecimal Notation: 0x0 to 0xFFFFFFFFFFFFFF

### 7-3.5.8 CIP Path

The CIP path string, having the type STRING, follows the format defined in this specification. The CIP path consists of groups of two adjacent hexadecimal characters separated by spaces. Double quotes enclose the entire string. Some example path strings are:

Path Ex.1: “20 04 24 01”

Path Ex.2: “20 05 24 02 30 04”

The CIP Path string may also include the keyword Param entries. Param entries in CIP Paths shall evaluate to a USINT, UINT or UDINT. The value of the Param shall be used in a little endian order for insertion into the path. The Param references within the path may be enclosed in brackets as shown in Path ex. 3. When enclosed in brackets, the value of the Param shall be local to the path – the same Param entry may have a different value elsewhere in the EDS. If the Param is not enclosed in brackets, the value shall be the same everywhere within the EDS.

Path Ex. 3: “20 04 24 [Param1] 30 03”

The user interface may ask the user to specify a value for Param1. If the user specifies “5” the resulting CIP path would be “20 04 24 05 30 03”.

The CIP Path string may also include the keyword, SYMBOL\_ANSI. When SYMBOL\_ANSI is encountered it shall evaluate to an extended symbolic segment (see Appendix C) entered through the user interface. The extended symbol segment shall be an ANSI extended symbol (CIP path type = 0x91). For example, the string "CAB" shall evaluate to the following extended symbol segment (padded): 0x91 0x03 0x43 0x41 0x42 0x00.

Path Ex. 4: “SYMBOL\_ANSI”

The user interface shall ask the user to specify a symbol string (like “CAB”) which results in a CIP Path of “91 03 43 41 42 00”.

### **7-3.5.9 STRINGI**

The CIP International String (STRINGI) data follows the format as defined in Volume I, Appendix C of the specification. The STRINGI is specified in an EDS file as a complex data representation. The entire content of the STRINGI specification is enclosed in a pair of braces. Each language specification within the STRINGI entry is also enclosed in a pair of braces. The members of each STRINGI entry are specified as 6 fields. The first three fields (the language selection) are expressed as exactly a three character fixed length string enclosed in double quotation marks. The data type EPATH is expressed in the string form of an EPATH (as described above). The character set selection is expressed as a UINT. The string content portion of the entry is expressed as a string or long string. For example, the following represents a STRINGI specification with three languages:

```
Field1 = { 3,
    {"eng",0xD0,4,"This is an ASCII English language string"},
    {"spa",0xD5,1000,L"Españoles palabras"}, $ "Spanish words" using UNICODE
    {"deu",0xD0,4, "Spanische Wörter auf Deutsch"} $ "Spanish words in German"
};
```

### **7-3.5.10 EDS Comments**

Comments must be delimited with the dollar sign character (\$) and the new line character. The EDS interpreter treats all characters between the comment delimiters as white space. Some example comments are:

\$ This is a valid comment line <NL>

1, 2, 3;

\$ This is a valid comment <NL>

\$ Comments cannot span <NL>  
more than one line <NL>

<= This is an error - no \$

### **7-3.5.11 EDS Date**

The EDS\_DATE data type shall be of the format; mm-dd-yyyy, where mm is the month, dd is the day of the month and yyyy is the year. Valid values for the month, day and year portions of the mm-dd-yyyy shall be

- mm 01 through 12
- dd 01 through 31 (depending upon the month and year)
- yyyy 1994 through 9999.

Two character years representations may be used in which case the EDS\_DATE data type shall be of the format; mm-dd-yy, where mm is the month, dd is the day of the month and yy is the year. In this case, the two digits for the year have an implied leading 19, such that yy=96 shall represent the year 1996. Valid values for the month, day and year portions of the mm-dd-yy parameters shall be:

- mm 01 through 12
- dd 01 through 31 (depending upon the month and year)
- yy 94 through 99 (Note that a leading 19 is implied)

NOTE: Two character year representations are not recommended.

### **7-3.5.12 EDS Time Of Day**

The EDS\_TIME\_OF\_DAY data type shall be of the format; hh:mm:ss, where hh is hours, mm is minutes and ss is seconds. Valid values for the hours, minutes and seconds shall be:

- hh 00 through 23
- mm 00 through 59
- ss 00 through 59

### **7-3.5.13 ASCII Floating Point Convention (REAL, LREAL)**

The REAL and LREAL data types represent binary floating-point values. The internal representation of these data formats is described by the IEEE Standard 754. This standard describes both numeric values and bit sequences which are interpreted as “Not a Number” (NaN) symbolic values and positive and negative infinity. The floating point values can be entered as either integer values, values based upon decimal floating point representation, or values entered in “scientific” notation using a base value and an offset in exponential form. Integer values are the same as those shown for the INT data type. These values can not be used to represent fractional values. Decimal floating-point values are those, which have both a whole integer and fractional component. The whole value and fractional components are separated by a decimal point “.” or period character. The exponential (scientific) notation form of the value is the same as the fractional value representation with the addition of an exponential component. This exponent is always a signed integer power to ten applied to the base value.

The range of legal REAL data (single IEEE, 32 bit format) is based upon the formula:

$$\text{value} = (-1)^s \cdot (2)^e \cdot (m)$$

Where:

- “s” is the value of the sign bit
- “e” is the eight bit exponent. This exponent allows an approximate exponent range between 0 and  $8.5e^{37}$ .
- “m” is the normalized 24-bit mantissa (23 internal to the storage plus one hidden bit). This allows a range of mantissa values to range between 0 and 8388607.

Since the mantissa of the floating-point representation only allows a value digit value to be precisely represented by the internal format, a large number of digits within the decimal (or mantissa) portion of a floating value is more for presentation convenience than precision. An arbitrary limit of 16 digits has been set for the CIP standard.

Integer (Fixed) Notation: -8388607 to 8388607

Decimal (Floating Point) Notation: 0.0 to  $\pm 9999999999999999$

Scientific Notation: 0.0 to  $\pm nn.nnnnnnnnE\pm xxxx$

Where the total number of digits in the mantissa (including the decimal point character and sign character) shall not exceed 13 characters and the number of characters in the exponent shall not exceed 6 characters (including the “E” character and sign character)

Where:

- “s” is the value of the sign bit
- “e” is the eleven-bit exponent. This exponent allows an approximate exponent range between 0 and  $8.5e^{37}$ .
- “m” is the normalized 53-bit mantissa (52 internal to the storage plus one hidden bit). This allows a range of mantissa values to range between 0 and 4503599627370495.

Since the mantissa of the floating-point representation only allows a value digit value to be precisely represented by the internal format, a large number of digits within the decimal (or mantissa) portion of a floating value is more for presentation convenience than precision. An arbitrary limit of 16 digits has been set for the CIP standard.

The range of legal LREAL data (double IEEE, 64 bit format) is based upon the formula:

$$\text{value} = (-1)^s \cdot (2)^{e-1023} \cdot (m)$$

Integer (Fixed) Notation: -4503599627370495 to 4503599627370495

Decimal (Floating Point) Notation: 0.0 to ±9999999999999999

Scientific Notation: 0.0 to ±nnnn.nnnnnnnnnnE±xxxx

Where the total number of digits in the mantissa (including the decimal point character and sign character) shall not exceed 18 characters and the number of characters in the exponent shall not exceed 6 characters (including the “E” character and sign character)

In addition to the above value entries, the floating-point representation allows for two styles of “Not a Number” or NaN symbolic entries and two forms of infinity. There are two types of NaN; a Signaling NaN and a Quiet NaN. Also, the format allows for the representation of the values positive and negative infinity. For these cases, the following special words are reserved and shall be used to represent the entry of the associated floating-point symbol:

- Quiet Not a Number: QUIET-NAN
- Signaling Not a Number: SIGNAL-NAN
- Positive Infinity: INFINITY (or +INFINITY)
- Negative Infinity: -INFINITY

#### **7-3.5.14 EDS\_URL Uniform Resource Locator**

All references to EDS\_URL within this specification are for the formalized information necessary to locate and access resources via an internet capable mechanism. The form of the EDS\_URL is that defined by the internet's Network Working Group RFC1738 "Uniform Resource Locator (URL)". For specifications made within an EDS file, the EDS\_URL is limited to any of the forms:

- http
- ftp
- file

## **7-3.6 Basic EDS File Requirements**

This section describes each section of an EDS and specifies the usage requirements.

**Table 7-3.2 Locating EDS Requirements**

<b>For EDS requirements for:</b>	<b>Go to section:</b>
Object Class Sections	7-3.6.1.1
Parameter Class Section	7-3.6.1.3
Connection Configuration Class Section	7-3.6.1.4
File Description Section	7-3.6.2
Device Description Section	7-3.6.3
Device Classification Section	7-3.6.4
Parameters Section	7-3.6.5
Parameter Groups Section	7-3.6.6
Assembly Section	7-3.6.7
Connection Manager Section	7-3.6.9
Port Section	7-3.6.10
Modular Section	7-3.7.1
Capacity Section	7-3.6.11
Event Enumeration Section	7-3.6.12
Symbolic Translation Section	7-3.6.13
Internationalization Section	7-3.6.14

### **7-3.6.1 Common Object Class Information**

#### **7-3.6.1.1 Object Class Sections**

The following table identifies section keywords that are used to publicize object class information. Each of the following EDS Section Keywords may contain any of the Common Object Class Entry Keyword(s) described in 7-3.6.1.2. The following table contains Object Class Section names.

**Table 7-3.3 Object Class Section Keywords**

<b>CIP Object</b>	<b>Section Keyword Name</b>	<b>Required/Optional</b>	<b>See section</b>
Identity Object 01 hex	[Identity Class]	Optional	
Message Router Object 02 hex	[Message Router Class]	Optional	
DeviceNet Object 03 hex	[DeviceNet Class]	Optional	
Assembly Object 04 hex	[Assembly Class]	Optional	
Connection Object 05 hex	[Connection Class]	Optional	
Connection Manager Object 06 hex	[Connection Manager Class]	Optional	

CIP Object	Section Keyword Name	Required/Optional	See section
Register Object 07 hex	[Register Class]	Optional	
Discrete Input Point Object 08 hex	[Discrete Input Class]	Optional	
Discrete Output Point Object 09 hex	[Discrete Output Class]	Optional	
Analog Input Point Object 0A hex	[Analog Input Class]	Optional	
Analog Output Point Object 0B hex	[Analog Output Class]	Optional	
Presence Sensing Object 0E hex	[Presence Sensing Class]	Optional	
Parameter Object 0F hex	[ParamClass]	Conditional	7-3.6.1.3
Parameter Group Object 10 hex	[Parameter Group Class]	Optional	
Group Object 12 hex	[Group Class]	Optional	
Discrete Input Group Object 1D hex	[Discrete Input Group Class]	Optional	
Discrete Output Group Object 1E hex	[Discrete Output Group Class]	Optional	
Discrete Group Object 1F hex	[Discrete Group Class]	Optional	
Analog Input Group Object 20 hex	[Analog Input Group Class]	Optional	
Analog Output Group Object 21 hex	[Analog Output Group Class]	Optional	
Analog Group Object 22 hex	[Analog Group Class]	Optional	
Position Sensor Object 23 hex	[Position Sensor Class]	Optional	
Position Controller Supervisor Object 24 hex	[Position Controller Supervisor Class]	Optional	
Position Controller Object 25 hex	[Position Controller Class]	Optional	
Block Sequencer Object 26 hex	[Block Sequencer Class]	Optional	
Command Block Object 27 hex	[Command Block Class]	Optional	
Motor Data Object 28 hex	[Motor Data Class]	Optional	
Control Supervisor Object	[Control Supervisor Class]	Optional	

CIP Object	Section Keyword Name	Required/Optional	See section
29 hex			
AC/DC Drive Object 2A hex	[AC/DC Drive Class]	Optional	
Acknowledge Handler Object 2B hex	[Acknowledge Handler Class]	Optional	
Overload Object 2C hex	[Overload Class]	Optional	
Softstart Object 2D hex	[Softstart Class]	Optional	
Selection Object 2E hex	[Selection Class]	Optional	
S-Device Supervisor Object 30 hex	[S-Device Supervisor Class]	Optional	
S-Analog Sensor Object 31 hex	[S-Analog Sensor Class]	Optional	
S-Analog Actuator Object 32 hex	[S-Analog Actuator Class]	Optional	
S-Single Stage Controller Object 33 hex	[S-Single Stage Class]	Optional	
S-Gas Calibration Object 34 hex	[S-Gas Calibration Class]	Optional	
Trip Point Object 35 hex	[Trip Point Class]	Optional	
Drive Data Object N/A	[Drive Data Class]	Optional	
File Object 37 hex	[File Class]	Optional	
S-Partial Pressure Object 38 hex	[S-Partial Pressure Class]	Optional	
ControlNet Object F0 hex	[ControlNet Class]	Optional	
ControlNet Keeper Object F1 hex	[ControlNet Keeper Class]	Optional	
ControlNet Scheduling Object F2 hex	[ControlNet Scheduling Class]	Optional	
Connection Configuration Object F3 hex	[Connection Configuration]	Conditional	7-3.6.1.4
Port Object F4 hex	[Port Class]	Optional	
TCP/IP Interface Object F5 hex	[TCP/IP Interface Class]	Optional	

CIP Object	Section Keyword Name	Required/Optional	See section
EtherNet Link Object F6 hex	[Ethernet Link Class]	Optional	

### 7-3.6.1.2 Common Object Class Entry Keywords

Every CIP object has certain characteristics that are described by the following common Object Class Section entry keywords. These keywords may be used in any Object Class Section (see chapter 7-3.6.1.1).

**Table 7-3.4 Common Object Class Entry Keywords**

Entry Name	Entry Keyword	Required/Optional
Revision	Revision	Required
Maximum Instance Number	MaxInst	Required
Number of Static Instances	Number_Of_Static_Instances	Required
Maximum Number of Dynamic Instances	Max_Number_Of_Dynamic_Instances	Required
Class attribute identification	Class_Attributes	Optional
Instance attribute identification	Instance_Attributes	Optional
Class service support	Class_Services	Optional
Instance service support	Instance_Services	Optional

#### 7-3.6.1.2.1 Revision Entry Keyword

The “Revision” entry keyword corresponds to the class attribute #1 (Revision). The “Revision” entry keyword shall contain the field shown in Table 7-3.5.

**Table 7-3.5 Revision Entry Keyword Format**

Field Name	Field Number	Data Type	Required/Optional
Revision	1	UINT	Required

##### 7-3.6.1.2.1.1 Revision, Field 1

This entry field contains the value that shall match the value of the object’s Revision class attribute (attribute #1).

#### 7-3.6.1.2.2 MaxInst Entry Keyword

The “MaxInst” entry keyword identifies the maximum instance number of the class. The “MaxInst” entry keyword shall contain the field shown in Table 7-3.6.

**Table 7-3.6 MaxInst Entry Keyword Format**

Field Name	Field Number	Data Type	Required/Optional
MaxInst	1	UINT	Required

#### **7-3.6.1.2.2.1 MaxInst, Field 1**

This entry field contains the value that identifies the maximum (numerically largest) instance number of the class.

#### **7-3.6.1.2.3 Number\_Of\_Static\_Instances Entry Keyword**

The “Number\_Of\_Static\_Instances” entry keyword identifies the total number of static instances of the object. The “Number\_Of\_Static\_Instances” entry keyword shall contain the field shown in Table 7-3.7.

**Table 7-3.7 Number\_Of\_Static\_Instances Entry Keyword Format**

Field name	Field Number	Data type	Required/Optional
Number of static instances	1	UINT	Required

#### **7-3.6.1.2.3.1 Number\_Of\_Static\_Instances, Field 1**

This entry field contains the value that identifies the total number of static instances of the object in the device.

#### **7-3.6.1.2.4 Max\_Number\_Of\_Dynamic\_Instances Entry Keyword**

The “Max\_Number\_Of\_Dynamic\_Instances” entry keyword identifies the maximum number of dynamic instances of the object capable of being created in the device. The “Max\_Number\_Of\_Dynamic\_Instances” entry keyword shall contain the field shown in Table 7-3.8.

**Table 7-3.8 Max\_Number\_Of\_Dynamic\_Instances Entry Keyword Format**

Field name	Field Number	Data type	Required/Optional
Maximum number of dynamic instances	1	UINT	Required

#### **7-3.6.1.2.4.1 Maximum Number\_Of\_Dynamic\_Instances Entry Keyword, Field 1**

This entry field contains the value that identifies the total number of dynamic instances of the object capable of being created in the device.

#### **7-3.6.1.2.5 Class\_Attributes Entry Keyword**

The “Class\_Attributes” entry keyword is used to identify attributes within an object class (instance number zero). The “Class Attributes” entry keyword shall contain the fields shown in Table 7-3.9.

**Table 7-3.9 Class\_Attributes Entry Keyword Format**

<b>Field name</b>	<b>Field Number</b>	<b>Data type</b>	<b>Required/Optional</b>
Attribute ID 1	1	UINT	Required
Attribute ID 2	2	UINT	Conditional
...			
Attribute ID n	N	UINT	Conditional

**7-3.6.1.2.5.1 Attribute ID Entry Keyword, Field 1 thru n**

These entry fields shall contain values that identify attributes of the object's class in the device. Each field in the list indicates an attribute that has been implemented in the object's class. The total number of fields is variable based upon the actual number of attributes implemented. No null fields are allowed.

**7-3.6.1.2.6 Instance\_Attributes\_Entry\_Keyword**

The “Instance\_Attributes” entry keyword is used to identify attributes within a class instance (non-zero instance number). The “Instance Attributes” entry keyword shall contain the fields shown in Table 7-3.10.

**Table 7-3.10 Instance\_Attributes Entry Keyword Format**

<b>Field name</b>	<b>Field Number</b>	<b>Data type</b>	<b>Required/Optional</b>
Attribute ID 1	1	UINT	Required
Attribute ID 2	2	UINT	Conditional
...			
Attribute ID n	N	UINT	Conditional

**7-3.6.1.2.6.1 Attribute ID Entry Keyword, Field 1 thru n**

These entry fields shall contain values that identify attributes of the object's instance(s) in the device. Each field in the list indicates an attribute that has been implemented in the object's instance(s). The total number of fields is variable based upon the actual number of attributes implemented. No null fields are allowed.

**7-3.6.1.2.7 Class and Instance Services Entry Keyword Fields**

The class and instance services keywords require a new field value type defined here. The name of this field type is Service

The following are the forms of the Service field:

- Service ID – Indication of non-attribute specific service ID code
- {Service ID, AttrId1, AttrId2, AttrId3, ...} – Indication of a service ID code and associates attribute identifiers to which the service is accepted.
- {Service ID, \*} – Indication of a service ID code and associates every attribute listed in the corresponding Class\_Attributes/Instance\_Attribute list.

### **7-3.6.1.2.7.1 Class\_Services Entry Keyword**

The “Class\_Services” entry keyword is used to identify services within an object class (instance number zero). The “Class Services” entry keyword shall contain the fields shown in Table 7-3.11.

**Table 7-3.11 Class\_Services Entry Keyword Format**

Field name	Field Number	Data type	Required/Optional
Service 1	1	Service	Required
Service 2	2	Service	Conditional
...			
Service n	N	Service	Conditional

#### **7-3.6.1.2.7.1.1 Class Services Entry Keyword, Field 1 thru n**

These entry fields shall contain values that identify services of the object’s class in the device. Each field in the list indicates a service that has been implemented in the object’s class. The total number of fields is variable based upon the number of services implemented. No null fields are allowed.

### **7-3.6.1.2.7.2 Instance\_Services Entry Keyword**

The “Instance\_Services” entry keyword is used to identify services within a class instance (non-zero instance number). The “Instance Services” entry keyword shall contain the fields shown in Table 7-3.12.

**Table 7-3.12 Instance\_Services Entry Keyword Format**

Field name	Field Number	Data type	Required/Optional
Service 1	1	Service	Required
Service 2	2	Service	Conditional
...			
Service n	n	Service	Conditional

#### **7-3.6.1.2.7.2.1 Instance Services Entry Keyword, Field 1 thru n**

These entry fields shall contain values that identify services of the object’s instance(s) in the device. Each field in the list indicates a service that has been implemented in the object’s instance(s). The total number of fields is variable based upon the number of services implemented. No null fields are allowed.

### **7-3.6.1.3 Parameter Class Section**

The parameter class section begins with the keyword [ParamClass]. The purpose of this section is to:

- identify the class level attributes of the configuration parameters
- contain a subset of the Parameter Object class attributes as defined in the CIP Object Library

The following table identifies the entry keywords in the parameter class section:

**Table 7-3.13 Entry Keywords in the Parameter Class Section**

Entry Name	Entry Keyword	Required/Optional
Revision	Revision	Optional <sup>2</sup>
Number of Static Instances	Number_Of_Static_Instances	Optional <sup>2</sup>
Maximum Number of Dynamic Instances	Max_Number_Of_Dynamic_Instances	Optional <sup>2</sup>
Parameter Class Descriptor	Descriptor	Required
Configuration Assembly Instance	CfgAssembly	Required
Safety Assembly Instance	SafetyCfgAssembly	Conditional <sup>1</sup>

1 - This keyword is not allowed if the device is not a safety device. If the device is a safety device, see Volume 5.

2 – This keyword, although defined in section 7-3.6.1.2 Common Object Class Entry Keywords as Required, is Optional in the [ParamClass] section.

---

- **Parameter Class Descriptor** - Contains bit flags that describe the behavior of the device's parameter objects.
- **Configuration Assembly Instance** - Identifies the configuration assembly object instance number for parameters associated in this EDS and defined in the CIP Object Library.
- **Safety Assembly Instance** – This entry keyword is a safety-specific keyword. Use of this entry keyword is described in the CIP Safety Volume

**Figure 7-3.6 Sample Electronic Data Sheet Illustrating the Parameter Class Section**

```
[File]
...
[Device]
...
[ParamClass]
    Revision = 1;
    MaxInst = 3;
    Number_Of_Static_Instances = 3;
    Max_Number_Of_Dynamic_Instances = 0;
    Descriptor = 0x0E;
    CfgAssembly = 3;

[Params]
...

[Groups]
...
```

### 7-3.6.1.4 Connection Configuration Class Section

The Connection Configuration section defines the characteristics of the Connection Configuration Object (see Object Library, Chapter 5) implemented in this device, if a Connection Configuration Object implementation exists. The Common Object Class keywords may be used in the Connection Configuration section, see chapter 7-3.6.1. The table below shows the additional entries in the Connection Configuration section. The Connection Configuration Class section begins with the keyword [Connection Configuration]. This section is required if the device supports the Connection Configuration object.

**Table 7-3.14 Entry Keywords in the Connection Configuration Section**

Entry Name	Entry Keyword	Required/Optional
Revision	Revision	Required
Number of Static Instances	Number_Of_Static_Instances	Optional <sup>1</sup>
Maximum Number of Dynamic Instances	Max_Number_Of_Dynamic_Instances	Optional <sup>1</sup>
Max Instances	MaxInst	Optional <sup>1</sup>
Mapping Attribute Format	MappingFormat	Required
Originator to Target Image Sizes and Paths	O2TImages	Required
Target to Originator Image Sizes and Paths	T2OImages	Required
Connection Target Name	ConnectionTarget	Optional
Redundant Owner Connection	RedundantOwnerConnections	Optional

<sup>1</sup> – This keyword, although defined in section 7-3.6.1.2 Common Object Class Entry Keywords as Required, is Optional in the [Connection Configuration] section.

### 7-3.6.1.4.1 Mapping Attribute Format Number Entry Keyword

The “**MappingFormat**” entry keyword shall contain the formatted field shown in Table 7-3.15.

**Table 7-3.15 MappingFormat Keyword Format**

Field name	Field Number	Data type	Required/Optional
Format	1	UINT	Required

#### 7-3.6.1.4.1.1 Format, Field 1

The format number of the mapping attribute (class attribute 8) supported by this device’s implementation of the Connection Configuration Object.

### 7-3.6.1.4.2 Originator to Target Image Sizes and Paths Entry Keyword

The “**O2TImages**” entry keyword shall contain the formatted field shown in Table 7-3.16.

**Table 7-3.16 O2TImages Keyword Format**

Field Name	Field Number	Data Type	Required/Optional
Reserved	1,2	empty	
Image Size	3,5,7,...	UINT	Field 3 is required. Fields 5, 7... are conditional based on the existence of additional image tables.
Image EPath	4,6,8,...	EPath	Optional

**7-3.6.1.4.2.1 Reserved, Field 1**

Field 1 shall be reserved for future definition and shall be empty.

**7-3.6.1.4.2.2 Reserved, Field 2**

Field 2 shall be reserved for future definition and shall be empty.

**7-3.6.1.4.2.3 Image Size**

The number of 8 bit units in the Originator to Target image table. Field 3 specifies the size of Originator to Target image table 1. Field 5 specifies the size of Originator to Target image table 2, etc.

**7-3.6.1.4.2.4 Image Path**

The EPath is a logical or symbolic segment used to access the Originator to Target image table. Field 4 specifies the EPath used to access the Originator to Target image table 1. Field 6 specifies the EPath used to access the Originator to Target image table 2, etc.

**7-3.6.1.4.3 Target to Originator Image Sizes and Paths Entry Keyword**

The "T2OImages" entry keyword shall contain the formatted field shown in Table 7-3.17.

**Table 7-3.17 T2OImages Keyword Format**

Field Name	Field Number	Data Type	Required/Optional
Reserved	1,2	empty	
Image Size	3,5,7,...	UINT	Field 3 is required. Fields 5, 7... are conditional based on the existence of additional image tables.
Image EPath	4,6,8,...	EPath	Optional

**7-3.6.1.4.3.1 Reserved, Field 1**

Field 1 shall be reserved for future definition and shall be empty.

**7-3.6.1.4.3.2 Reserved, Field 2**

Field 2 shall be reserved for future definition and shall be empty.

#### **7-3.6.1.4.3.3 Image Size**

The number of 8 bit units in the Target to Originator image table. Field 3 specifies the size of Target to Originator image table 1. Field 5 specifies the size of Target to Originator image table 2, etc.

#### **7-3.6.1.4.3.4 Image EPath**

The EPath is a logical or symbolic segment used to access the Target to Originator image table. Field 4 specifies the EPath used to access the Target to Originator image table 1. Field 6 specifies the EPath used to access the Target to Originator image table 2, etc.

#### **7-3.6.1.4.4 Connection Target Name Entry Keyword**

Each entry keyword shall be “**ConnectionTarget**” combined with a number (decimal), for example, “**ConnectionTarget1**”. The number (decimal) shall be used to locate the corresponding “**Connection**” entry in the Connection Manager section of this device. One Connection Target Name entry may exist for each Connection entry specified in the Connection Manager section. If no ConnectionTarget entry corresponding to a Connection entry exists, the name from the Connection entry shall be used for the target entry name. The “**ConnectionTargetName**” keyword shall contain the formatted field shown in Table 7-3.18.

**Table 7-3.18 ConnectionTarget Entry Keyword Format**

Field Name	Field Number	Data Type	Required/Optional
Name	1	Eds_Char_Array	Required
International Name	2	STRINGI	Optional

##### **7-3.6.1.4.4.1 Name, Field 1**

The name to display for the target entry that corresponds to the connection specified in the Connection Manager section by a Connection entry.

##### **7-3.6.1.4.4.2 International Name, Field 2**

The connection target name expressed in STRINGI notation.

#### **7-3.6.1.4.5 Redundant Owner Connection Support Entry Keyword**

The “**RedundantOwnerConnections**” entry keyword is optional and shall contain the formatted field shown in Table 7-3.19. If this optional keyword is omitted, redundant connections can not be originated.

**Table 7-3.19 RedundantOwnerConnections Keyword Format**

Field name	Field Number	Data type	Required/Optional
Supported	1	Field Keyword, possible values: Yes, No	Required

### **7-3.6.1.4.5.1 Supported, Field 1**

This field indicates if redundant owner connections can be originated. If the field value is “Yes”, redundant connections can be originated. If the field value is “No”, redundant connections can not be originated. When redundant owner connections can be originated, as few as zero of the connections being originated, and as many as all the connections being originated may be redundant owner connections.

### **7-3.6.1.4.5.1.1 Example Connection Configuration Section**

The following is an example of a [Connection Configuration] section:

**Figure 7-3.7 Sample Electronic Data Sheet Illustrating the Connection Configuration Class Section**

```
$ The following EDS section describes a device that supports the 3rd
$ revision of Connection Configuration Object. The following are the
$ characteristics of this Connection Configuration Object implementation:
$
$ The maximum number of Connection Configuration Object instances
$ (connections or targets) that can be created is 128.
$ The Originator to Target Image (output) size is 4000 bytes long. The
$ Originator to Target Image can be read by performing a Get_Member
$ service to instance one of the Assembly Object (attribute 3).
$ The Target to Originator Image (input) size is 4000 bytes long. The
$ Target to Originator Image can be read by performing a Get_Member
$ service to instance two of the Assembly Object (attribute 3).
$ The name of a connection to this device is "Consume Data From", the
$ connection path is "20 04 24 01 2C 01 2C 02". If a target for a
$ connection to this device is created it shall be called "Send Data".
$ This device does not support origination of Redundant Owner connections.
$ The Connection Configuration Object Open_Connection, Close_Connection
$ and Stop_Connection instance services are supported.
$ The ability to change this devices mode is supported by using the
$ Set_Attribute_Single service to the Connection Configuration Object
$ class attribute 0x33.

[File]
...
[Device]
...
[Connection Configuration]
    Revision = 3;           $ revision of the Connection Configuration object
                           $ supported in this device
    MappingFormat = 0;      $ Format of mapping attribute
                           $ (class attribute 8)
    MaxInst = 128;         $ maximum number of connection configuration
                           $ object instances supported
    O2TImages = ,,4000,"20 04 24 01 30 03";
                           $ 4000 bytes of output image, path of output
                           $ image table
    T2OImages = ,,4000, "20 04 24 02 30 03";
                           $ 4000 bytes of input image, path of input
                           $ image table
    ConnectionTarget1 = "Send Data";
                           $ target name corresponding to Connection1
```

```

RedundantOwnerConnections = No;
                           $ redundant output owner connections not
                           $ supported
Number_Of_Static_Instances = 0;
Max_Number_Of_Dynamic_Instances = 128;
Instance_Services = 0x4C,           $ Open_Connection,
                    0x4D,           $ Close_Connection,
                    0x4E;          $ Stop_Connection,
                           $ services to instances supported
Class_Services = {0x10, 33};      $ Set_Attribute_Single service to
                           $ attribute
                           $ 0x33 supported (set scanner mode)
...
[Assembly Class]
Revision = 2;
MaxInst = 177;
Number_Of_Static_Instances = 14;
Max_Number_Of_Dynamic_Instances = 0;
Instance_Services = 0x18;          $ Get_Member service to instances
                           $ supported (read of input and output
                           $ image tables supported)
...
[Connection Manager]
Connection1 =
0x02010002,                      $ trigger and transport
0x44244305,                      $ connection parameters
,0,,                            $ O->T RPI, size and format
,4,,                            $ T->O RPI, size and format
,,
,,
"Consume Data From",            $ config part 1
",",                            $ config part 2
"Consume Data From",            $ connection name
",",                            $ help string
"20 04 24 01 2C 01 2C 02";    $ connection path

```

### 7-3.6.2 File Description Section

The file description section begins with the keyword [File] and contains administrative information about the EDS file. A configuration tool reads this information, formats it, and displays it to the user. The user can also access this section with a text file viewer and display the unformatted information. This section requires no modification unless the user manually modifies the file. The file description section must contain:

**Table 7-3.20 Fields in the File Description Section**

Entry Name	Entry Keyword	Field Number	Data Type	Required/Optional
File Description Text	DescText	1	ASCII Character Array	Required
File Creation Date	CreateDate	1	DATE	Required
File Creation Time	CreateTime	1	TIME_OF_DAY	Required
Last Modification Date	ModDate	1	DATE	Conditional
Last Modification Time	ModTime	1	TIME_OF_DAY	Conditional
EDS Revision	Revision	1	REVISION	Required
Home URL	HomeURL	1	ASCII Character Array	Optional
Exclude	Exclude	1	ASCII Character Array	Optional

Entry Name	Entry Keyword	Field Number	Data Type	Required/Optional
EDS File CRC	EDSFileCRC		See CIP Safety Specification (Volume 5, Chapter 7)	Conditional 1

1- This keyword is not allowed if the device is not a safety device. If the device is a safety device, see Volume 5.

---

The entries in the file description section provide:

- **File Description Text** - a single line of text displayed by the configuration tool. The EDS developer assigns a meaningful line of text for this entry. Double quotes enclose all character arrays.
- **File Creation Date** - the creation date of the EDS, assigned by the EDS developer. Provided only for convenience, you can use this date to get version information about the file. A configuration tool does not use this information to perform any type of version control, but it may display the contents.
- **File Creation Time** - the creation time of the EDS, assigned by the EDS developer. Provided only for convenience, you can use this date to get version information about the file. A configuration tool does not use this information to perform any type of version control, but it may display the contents.
- **Last Modification Date** - the date of the last modification to the EDS. A configuration tool that allows modification of the EDS file shall update this field as needed. Provided only for convenience, the configuration tool displays the contents of this entry if it exists. If a configuration tool changes the EDS, the configuration tool shall update this field. However, if you modify the EDS manually or with a text editor, you should also update this field.

This keyword is required if either:

- the EDS file is modified by a software tool, or
- the Last Modification Time keyword is present
- **Last Modification Time** - the time of the last modification to the EDS. A configuration tool that allows modification of the EDS file must update this entry as needed. Provided for convenience, the configuration tool displays the contents of this entry if it exists. If a configuration tool changes the EDS, the configuration tool must update this field. However, if you modify the EDS manually or with a text editor, you should also update this field.
- **EDS Revision** - the revision of the EDS. The EDS revision is not required to have any relationship to the product's revision, it is simply the revision of the EDS file itself. The revision must be formatted as:

major\_revision.minor\_revision

For example: 2.1      <= major revision is 2, and minor revision is 1

- **Home URL** – Uniform Resource Locator of the master EDS file, the Icon file and other files related to this EDS. The Home URL shall specify a complete qualified URL for referencing a master version of the EDS file. In addition, the referenced area (without the file name specification) is used to specify an area where other related file(s) relating to the device described by this EDS are contained.

- **Exclude** – This keyword, when present, indicates that the manufacturer of the device expects a specific behavior by the configuration tool if that tool encounters any items (section keywords, entry keywords, field values, etc), which are not understood. The behavior modification of the tool is defined as which type of operations shall be excluded when any items, which are not understood are encountered. This keyword has a single required field with the following allowed values: “NONE”, “WRITE”, “READ\_WRITE”. Each word shall be interpreted as:
  - **NONE** – No behaviors are excluded.
  - **WRITE** – All write access to the device shall be excluded (e.g. no Set\_Attribute\_Single, Set\_Member, etc. service shall be performed). The configuration tool shall also exclude configuring a second device, which results in the second device writing to the device described by this EDS.
  - **READ\_WRITE** – All access to the device is excluded (e.g. no Get\_Attribute\_Single, Get\_Member, etc. service shall be performed). The configuration tool shall also exclude configuring a second device, which results in the second reading from or writing to the device described by this EDS.

If the Exclude keyword does not exist in the EDS file, Exclude = NONE is the default.

When a software tool prevents writing or reading/writing a device due to the Exclude keyword messages shall be displayed by the software tool indicating why the operation is excluded. The following text strings shall be used:

- When excluding writing a device because the software tool does not understand a vendor specific item (section keywords, entry keywords, field values, etc) – “This device cannot be configured because the device supports some vendor-specific behavior that is not supported by this software.”
- When excluding writing a device because the software tool does not understand an open item (section keywords, entry keywords, field values, etc) – “This device cannot be configured because this software does not support a CIP defined entry described in this devices’ EDS file. Contact the vendor of this software for an updated version.”
- When excluding reading/writing a device because the tool does not understand a vendor specific item (section keywords, entry keywords, field values, etc.) – “The configuration for this device cannot be displayed because the device supports some vendor-specific behavior that is not supported by this software.”
- When excluding reading/writing a device because the tool does not understand an open item (section keywords, entry keywords, field values, etc.) – “The configuration for this device cannot be displayed because this software does not support a CIP defined entry described in this devices’ EDS file. Contact this vendor of the software for an updated version.”
- When multiple conditions apply, (both vendor specific and open items are not understood) the tool shall display both text strings.

**Figure 7-3.8 Sample Electronic Data Sheet Illustrating the File Section**

```
[File]
DescText = "Smart Widget EDS File";
CreateDate = 04-03-1994;           $ created
CreateTime = 17:51:44;
ModDate = 04-06-1994;           $ last changed
ModTime = 22:07:30;
Revision = 2.1;                  $ Revision of EDS
HomeURL = "http://www.odva.org/EDS/example.eds";

[Device]
...
[ParamClass]
...
[Params]
...
[Groups]
...
```

### 7-3.6.3 Device Description Section

The Device Description section begins with the keyword [Device] and contains a manufacturer's information about the device, including some of the same values in a device's Identity Object. The table below shows the entries in the device description section.

**Table 7-3.21 Fields in the Device Description Section**

Entry Name	Entry Keyword <sup>1</sup>	Field Number	Data Type	Required/Optional
Vendor ID <sup>2,3</sup>	VendCode	1	UINT	Required
Vendor Name	VendName	1	ASCII Character Array	Required
Device Type <sup>2,3</sup>	ProdType	1	UINT	Required
Device Type String	ProdTypeStr	1	ASCII Character Array	Required
Product Code <sup>2,3</sup>	ProdCode	1	UINT	Required
Major Revision <sup>2,3</sup>	MajRev	1	USINT	Required
Minor Revision <sup>2</sup>	MinRev	1	USINT	Required
Product Name <sup>2</sup>	ProdName	1	ASCII Character Array	Required
Catalog Number	Catalog	1	ASCII Character Array	Optional
Exclude from Adapter Rack Connection	ExcludeFromAdapterRackConnection	1	ASCII Character Array	Optional
Icon File Name	Icon	1	ASCII Character Array	Optional
Device Status Assembly	DeviceStatusAssembly	1	UINT	Optional

1 - The SerNum and Comment entry keywords have been obsoleted

2 - This entry represents an attribute of the Identity Object

3 - This entry is used to match an EDS with a specific product/revision

The entry name for the device description field describes the unique data entry line number. A configuration tool uses the required entries in the device description section to match the EDS to the device being configured. The entries in the device description section provide:

- **Vendor ID** - Numeric vendor identifier as defined by the Identity Object, Attribute 1.

- **Vendor Name** - Textual vendor name. When displayed, truncation may occur to meet the display capabilities.
- **Device Type** - Numeric device identifier as defined by the Identity Object, Attribute 2.
- **DeviceType String** - Textual description of device type exactly as shown in CIP Common, Chapter 6-7. The string for vendor specific device types is at the vendors' discretion.
- **Product Code** - Vendor assigned numeric product code identifier as defined by the Identity Object, Attribute 3. Each product code shall have its own EDS.
- **Major Revision** - Vendor-assigned major revision number as defined by the Identity Object, Attribute 4. The major revision of a product is typically incremented when there is a change to the form, fit, or function of the device. Changes to major revisions are used by a configuration tool to match a device to an EDS.
- **Minor Revision** - Vendor-assigned minor revision number as defined by the Identity Object, Attribute 4. The minor revision number is used to identify changes in a product that do not effect user configuration choices. For example: firmware bug fixes, an additional LED, internal hardware changes, etc. Changes in minor revisions are not used by a configuration tool to match a device with an EDS.
- **Product Name** - Textual product name as defined by the Identity Object, Attribute 7. When displayed, truncation may occur to meet the display capabilities.
- **Catalog Number** - Textual catalog or model number. One or more catalog numbers may be associated with a particular product code. In the case of multiple catalog numbers, it is still useful to provide as much of the catalog number as is practical. For example, 1438-BAC7xx where 'xx' represents variants in the catalog number supported by this product code/EDS.
- **ExcludeFromAdapterRackConnection** - This field is used to describe if a rack-based device must be excluded from an adapter rack connection. If the field value is the string "Yes" this module shall be excluded from adapter rack connections by resetting the associated slot mask bits (input, output and configuration). If the field value is the string "No" or this optional field is omitted the associated slot mask bits may be set.
- **Icon File Name**- File name of an Icon file. Identifies a file that contains a graphical representation of the device. The file shall have the \*.ICO MSWindows format, and shall minimally contain a 16x16 icon. The file may also contain 32x32, 48x48, and 64x64 icons. The location of the Icon file is the combination of the location specified by the HomeURL keyword (without the HomeURL file name component) and the file name specified by this keyword. This keyword shall only be present when a HomeURL keyword exists.
- **Device Status Assembly** – Identifies the value (N) of the AssemN that defines the format and enumerations of the Identity Object status attribute.

**Figure 7-3.9 Sample Electronic Data Sheet Illustrating the Device Description Section**

```
[File]
...
[Device]
VendCode = 65535;
VendName = "Widget-Works, Inc.";
ProdType = 0;
ProdTypeStr = "Generic";
ProdCode = 42;
MajRev = 1;                      $ Device Major Revision
MinRev = 1;                      $ Device Minor Revision
ProdName = "Smart-Widget";
Catalog = "1499-DVG";
Icon = "example.ico";
[ParamClass]
...
[Params]
...
[Groups]
...
```

#### **7-3.6.4 Device Classification Section**

The Device Classification section begins with the keyword [Device Classification] and shall classify the device described by the EDS into one or more categories of devices. The entry keyword for all classifications shall consist of the character array, "Class", combined with a decimal number. The numbers shall start at 1 for the first class, and shall be incremented for each additional class. Commas shall separate all fields, and a semicolon shall indicate the end of the entry.

The number of fields for each classification entry shall be variable to allow a tree classification structure similar to a file systems directory structure. Sub-classification of the public classifications shall be reserved. Vendor-specific classifications may be sub-classified at the discretion of the vendor. The first field shall represent the highest level in the tree structure and shall be one of the following:

- **CompoNet;**
- ControlNet;
- DeviceNet;
- EtherNetIP
- **ModbusSL**
- **ModbusTCP**
- a vendor-specific field.

The vendor-specific field shall begin with the Vendor ID of the company making the addition followed by an underscore (*VendorID\_VendorSpecificField*). The VendorID shall be displayed in decimal and shall not contain leading zeroes. Each vendor is responsible for maintaining and documenting their vendor-specific field.

### 7-3.6.5 Parameters Section

The parameters section begins with the keyword [Params] and identifies all of the configuration parameters in a device. All configuration performed on a device depends on the configuration parameters.

The CIP Parameter Object definition provides more specific details about the parameter fields, including the allowed data types and lengths. In general, the EDS parameter fields follow the definitions of the CIP Parameter Object instance attributes, except as noted below.

The parameter section shall identify the configuration parameters in a device. The entry keyword shall be one of the following character arrays, “Param”, “ProxyParam”, “ProxiedParam”, combined with the Parameter object instance number (decimal) for the device, e.g. “Param1”. When a parameter object instance exists within a node, and if this parameter is also described within an EDS, then the value of “N” in “ParamN” shall be equal to the parameter object instance. The actual parameter object instance may, but need not be, implemented in the device. Conversely, it is not required that ALL parameter object instances have a corresponding “ParamN” entry in an EDS. Commas shall separate all fields, and a semicolon shall indicate the end of an entry. Each entry contains the formatted fields shown in the table below. The “ProxyParam” and “ProxiedParam” keywords are defined further in Chapter 7-3.7, Modular EDS File Requirements.

**Table 7-3.22 Fields in the Parameters Section**

Field Name	Field Number	Data Type	Required/Optional
Reserved	1	USINT	Required
Link Path Size	2	USINT	Required
Link Path	3	EPAUTH	Required
Descriptor	4	WORD	Required
Data Type	5	EPAUTH	Required
Data Size	6	USINT	Required
Parameter Name	7	ASCII Character Array	Required
Units String	8	ASCII Character Array	Required
Help String	9	ASCII Character Array	Required
Minimum Value	10	data type	Conditional <sup>1</sup>
Maximum Value	11	data type	Conditional <sup>1</sup>
Default Value	12	data type	Required
Scaling Multiplier	13	UINT	Optional
Scaling Divider	14	UINT	Optional
Scaling Base	15	UINT	Optional
Scaling Offset	16	DINT <sup>2</sup>	Optional
Multiplier Link	17	UINT	Optional
Divisor Link	18	UINT	Optional
Base Link	19	UINT	Optional
Offset Link	20	UINT	Optional
Decimal Precision	21	USINT	Optional

Field Name	Field Number	Data Type	Required/Optional
International Parameter Name	22	STRINGI	Optional
International Engineering Units	23	STRINGI	Optional
International Help String	24	STRINGI	Optional

1 See Maximum/Minimum Attribute Semantics in the Parameter Object definition in Volume 1, Chapter 5 for further information.

2 The data type for this field deviates from the Parameter Object, which is specified as an INT. If the ParamN entry is referencing a parameter that exists in a Parameter Object within a device, this value will need to be kept in the INT range.

The following ParmN fields, which require additional definition (beyond the corresponding Parameter Object instance attribute definition), are listed below.

- **Reserved** - This first field shall contain a zero.
- **Link Path** - The link path identifier. The path shall be entered as a character array, using the path notation described in Section 7-3.5.8. If the Not Addressable descriptor bit value is 1 or the Link Path is empty, the attribute described by this ParamN is not directly addressable from the network. If the Not Addressable descriptor bit value is 0 and this field is a null string, “”, it shall be addressable as the data attribute (instance attribute 1) of the Nth instance of the Parameter object.
- **Parameter Name** - The textual parameter name. This field may be longer than the limit specified in the Parameter Object. A tool may truncate this text to the limit specified in the Parameter Object.
- **Units String** - The textual display units character array. This field may be longer than the limit specified in the Parameter Object. A tool may truncate this text to the limit specified in the Parameter Object.
- **Help String** - The textual help character array. This field may be longer than the limit specified in the Parameter Object. A tool may truncate this text to the limit specified in the Parameter Object.

The “**Enum**” keyword shall be used to provide an enumeration list of parameter choices to present to the user. The entry keyword for all enumerated parameters shall consist of the character array, “**Enum**”, combined with the decimal number from the corresponding Param entry. Commas shall separate all fields, and a semicolon shall indicate the end of the entry. Each Enum entry shall consist of pairs of integers and strings.

**Figure 7-3.10 Sample Electronic Data Sheet Illustrating the Parameters Section**

```
[File]
...
[Device]
...
[ParamClass]
...
[Params]
Param1 = 0, 1,"20 02", 0x0014, 0xD2, 2, "Preset", "V",
        "User Manual p33", 0, 5, 1, 1, 1, 1, 0, 0, 0, 0, 0, 2;

Param2 =                      $ parameter instance
0,                          $ First field shall equal 0
6, "20 04 24 01 30 03",    $ path size, path
0x0014,                     $ descriptor - in hex format
```

## 7-3.6.6 Parameter Groups Section

The parameter groups section begins with the keyword [Groups] and identifies all of the parameter groups in a device. Each parameter group contains a list of the parameter object instances in the group. The entry keyword for each group consists of the combination of the character array, “**Group**”, and the Parameter Group instance number (decimal) in the device, for example “**Group1**”. The decimal numbers must start at one and increment by one.

The fields in each entry contain the group name, the number of members in the group, and the instance numbers of the parameter objects in the group or **VariantN/VariantExaN** where N is the variant number in this EDS or ProxyN where N is the ProxyParam number in this EDS. A comma separates all fields, and a semicolon indicates the end of the entry. The parameter group section contains:

**Table 7-3.23 Fields in the Parameter Groups Section**

Field Name	Field Number	Data Type	Required/Optional
Group Name String	1	ASCII Character Array	Required
Number of Members	2	UINT	Required
Parameter, Proxy Parameter or Variant	3 thru (number of members + 2)	UINT for Param, ProxyN for Proxy Param or VariantN/VariantExaN for Variant	Required

**Figure 7-3.11 Sample Electronic Data Sheet Illustrating the Parameter Groups Section**

```
$  
[File]  
...  
[Device]  
...  
[ParamClass]  
...  
[Params]  
...  
[Groups]  
Group1 = "Setup", 3, 1, Variant2, Proxy1; $ group 1 named Setup consists  
$ of Param1, Variant2 & ProxyParam1  
  
Group2 = "Monitor", 3, 2, 3, Proxy4; $ group 2 named Monitor consists of
```

```
$ Param2, Param3 and ProxyParam4  
Group3 = "Maintenance", 2, 1, 3;      $ group 3 named Maintenance  
                                         $ consists of Param1 and Param3
```

### **7-3.6.7 Assembly Section**

The Assembly section begins with the keyword [Assembly] and describes the structure of a data block. Often this block is the data attribute of an Assembly object; however, this section of the EDS can be used to describe any complex structure. The description of this data block parallels the mechanism that the Assembly object uses to describe its member list.

#### **7-3.6.7.1 Revision Entry Keyword**

The "Revision" entry keyword shall have one 16-bit integer field that shall be the revision (class attribute 1) of the Assembly object within the device. If this optional entry is missing, the revision of the Assembly object shall be 2.

#### **7-3.6.7.2 Assem type Entry Keywords**

The following character arrays are the roots for Assem type entry Keywords:

- Assem
- AssemExa
- ProxyAssem
- ProxyAssemExa
- ProxiedAssem
- ProxiedAssemExa

The Assem type entry keyword shall consist of a root from the list above combined with the Assembly object instance number (decimal) for the device, e.g. "**Assem1**". The same instance number (decimal) shall not be used for an "**AssemN**" and an "**AssemExaN**" entry. The same instance number (decimal) shall not be used for a "**ProxyAssemN**" and a "**ProxyAssemExaN**" entry. The same instance number (decimal) shall not be used for a "**ProxiedAssemN**" and a "**ProxiedAssemExaN**" entry. If a particular instance of the Assembly object is addressable from the link, there shall be a one-for-one pairing between the **Assem** number in the EDS file and the Assembly instance number in the device. Commas shall separate all fields, and a semicolon shall indicate the end of the entry. A white space or nothing between commas shall be used for optional fields not provided. The "**ProxyAssemN**", "**ProxyAssemExaN**", "**ProxiedAssemN**", and "**ProxiedAssemExaN**" keywords are defined further in Chapter 7-3.7, Modular EDS File Requirements.

"**AssemExaN**" is an extension of the "**AssemN**" keyword. "**ProxyAssemExaN**" is an extension of the "**ProxyAssemN**" keyword. "**ProxiedAssemExaN**" is an extension of the "**ProxiedAssemN**" keyword. Each "**AssemN**", "**ProxyAssemN**" and "**ProxiedAssemN**" entry shall contain the formatted fields shown in Table 7-3.24. Each "**AssemExaN**", "**ProxyAssemExaN**" and "**ProxiedAssemExaN**" entry shall contain the formatted fields shown in Table 7-3.25.

**Table 7-3.24 AssemN Keyword Format**

Field Name	Field Number	Data Type	Required/Optional
Name	1	Eds_Char_Array	Optional
Path	2	EPATH	Optional
Size	3	UINT	Optional
Descriptor	4	WORD	Optional
Reserved	5, 6	empty	Optional
Member Size	7, 9, 11 ...	UINT	Conditional
Member Reference	8, 10, 12 ...	Empty, AssemN, ParamN, Number or EPATH	Conditional

**Table 7-3.25 AssemExaN Keyword Format**

Field name	Field number	Data type	Required/ Optional
Name	1	Eds_Char_Array	Optional
Path	2	EPATH	Optional
Size	3	UINT or ParamN	Optional
Descriptor	4	WORD	Optional
Reserved	5, 6	empty	Optional
Member Size	7, 9, 11 ...	UINT	Conditional
Member Reference	8, 10, 12 ...	AssemN, AssemExaN, ParamN, Number, VariantN, VariantExaN or EPATH	Conditional

### 7-3.6.7.2.1 Name, Field 1

The first field, called "Name", shall be a string giving a name to the data block. This optional field may be used by a user interface.

### 7-3.6.7.2.2 Path, Field 2

The second field, called "Path", shall be a string that specifies a logical path. This path shall identify the address of the data block within the device. If the Not Addressable descriptor bit value is 1 or this field is empty, the block described by this **AssemN** entry is not directly addressable from the link. If the Not Addressable descriptor bit value is 0 and this field is a null string, "", the data block shall be addressable as the data attribute (instance attribute 3) of the Nth instance of the Assembly object.

### 7-3.6.7.2.3 Size, Field 3

The third field, called "Size", shall be the size of the data block in bytes. When the "Size" field is specified as a ParamN, the type of the ParamN shall be UINT or USINT. When the "Member Size"/"Member Reference" fields are present and the "Size" field is present, if the "Size" field is smaller than the defined size of the corresponding "Member Size"/"Member Reference" fields, the least significant bits of the corresponding "Member Reference" fields shall be used. When the "Member Size"/"Member Reference" fields are present and the "Size" field is present, if the "Size" field is larger than the defined size of the corresponding "Member Size"/"Member Reference" fields, the entire "Member Reference" fields shall be followed by zero pads to extend the assembly to the "Size".

When the “Member Size”/“Member Reference” fields are present and the “Size” field is not present, the defined size of the corresponding “Member Size”/“Member Reference” fields shall be used.

When the “Member Size”/“Member Reference” fields are not present and the “Size” field is not present, the size shall be 0.

#### 7-3.6.7.2.4 Descriptor, Field 4

The fourth field, “Descriptor”, shall be a bit-field that describes certain properties of the Assembly. The bits of this field are described in Table 7-3.26. If this field is empty it shall be interpreted as 0.

**Table 7-3.26 Descriptor Field Bit Value**

Bit	Name	Value	Meaning
0	Allow Value Edit	0	The contents of value member references may not be edited. The member references considered to be values are a Number, an empty field and a string representing an EPATH.
		1	The contents of value member references may be edited. The member references considered to be values are a Number, an empty field and a string representing an EPATH.
1-3	Assem Type	0	Not modular Assem
		1	This Assem defines a fixed slot size, expanded adapter rack Assem, the adapter device is not included in the Assem.  This Assem applies to each slot in the rack, in slot order. This Assem shall specify a UINT size field, this UINT size shall be the amount of data supplied, in bytes, for each slot in the rack. Each module participating in the adapter rack Assem shall supply the same amount of data. For empty slots and slots containing modules not participating in this adapter rack Assem the pad data shall be included, the size of the pad data shall be specified by the UINT size field. The member list size may be less than or equal to the size field value. If the member list size is less than the size field value the Assem shall be padded to the size field value.
		2	This Assem defines a fixed slot size, compressed adapter rack Assem, the adapter is not included in the Assem.  This Assem applies to each slot in the rack, in slot order, that is participating in this adapter rack Assem. This Assem shall specify a UINT size field, this UINT size shall be the amount of data supplied, in bytes, for each slot in the rack participating in the adapter rack Assem. The member list size may be less than or equal to the size field value. If the member list size is less than the size field value the Assem shall be padded to the size field value.
		3	This Assem defines a variable sized, compressed adapter rack Assem, the adapter is not included in the Assem.  This Assem applies to each slot in the rack, in slot order, that is participating in this adapter rack Assem. This Assem may specify a ProxyParamN size field which allows each module EDS to define the amount of data included in this adapter rack Assem.
		4	Reserved.

Bit	Name	Value	Meaning
		5	<p>This Assem defines a fixed slot size, expanded adapter rack Assem, the adapter device is included in the Assem.</p> <p>This Assem applies to each slot in the rack, in slot order. This Assem shall specify a UINT size field, this UINT size shall be the amount of data supplied, in bytes, for each slot in the rack. Each module participating in the adapter rack Assem shall supply the same amount of data. For empty slots and slots containing modules not participating in this adapter rack Assem the pad data shall be included, the size of the pad data shall be specified by the UINT size field. The member list size may be less than or equal to the size field value. If the member list size is less than the size field value the Assem shall be padded to the size field value.</p>
		6	<p>This Assem defines a fixed slot size, compressed adapter rack Assem, the adapter is included in the Assem.</p> <p>This Assem applies to each slot in the rack, in slot order, that is participating in this adapter rack Assem. This Assem shall specify a UINT size field, this UINT size shall be the amount of data supplied, in bytes, for each slot in the rack participating in the adapter rack Assem. The member list size may be less than or equal to the size field value. If the member list size is less than the size field value the Assem shall be padded to the size field value.</p>
		7	<p>This Assem defines a variable sized, compressed adapter rack Assem, the adapter is included in the Assem.</p> <p>This Assem applies to each slot in the rack, in slot order, that is participating in this adapter rack Assem. This Assem may specify a ProxyParamN size field which allows each module EDS to define the amount of data included in this adapter rack Assem.</p>
4	Not Addressable	0	If the Path field is not empty, the block described by this assembly entry is directly addressable from the link via the Set_Attribute and Get_Attribute services.
		1	The block described by this assembly entry is not directly addressable from the link via the Set_Attribute or Get_Attribute services.
5-7	NULL Indirect Parameter Handling	0	If a member reference is a Parameter whose Indirect Parameter Reference bit is set and whose referenced parameter value is 0 (NULL), then the member size and member reference pair is ignored.
		1	If a member reference is a Parameter whose Indirect Parameter Reference bit is set and whose referenced parameter value is 0 (NULL), then the member size and member reference pair and any following member size and member reference pairs are ignored.
		2	If a member reference is a Parameter whose Indirect Parameter Reference bit is set and whose reference parameter value is 0 (NULL), then the number of bits specified by the member size field shall be used as a pad within the Assembly object.
		3-7	Reserved
8-15	Reserved	0	

See 7-3.6.7.4 Examples of Adapter/Rack Assem Data Formatting for examples.

#### **7-3.6.7.2.5 Reserved Field 5**

Field 5 shall be reserved for future definition and shall be empty.

#### **7-3.6.7.2.6 Reserved Field 6**

Field 6 shall be reserved for future definition and shall be empty.

### **7-3.6.7.2.7 Member Size/Member Reference Fields**

The remaining fields shall be paired such that a "Member Size" field is paired with a "Member Reference" field making the total number of fields even. Zero remaining fields is valid, which represents an empty assembly. The pairs shall correspond to an Assembly object member list.

The allowed entries for the "Member Reference" field shall be one of the following:

- a ParamN entry from the [Params] section;
- an AssemN entry from the [Assembly] section;
- an AssemExaN entry from the [Assembly] section;
- a VariantN entry from the [Assembly] section;
- a VariantExaN entry from the [Assembly] section;
- a string representing a path (EPATH);
- a number, maximum value  $2^{32}-1$ ;
- an empty field;

If the "Member Reference" field is empty, the number of bits specified by the "Member Size" field shall be used as a pad within the Assembly object. A "Member Reference" field containing a null string shall be treated as if the field was empty. A "Member Reference" field and its corresponding "Member Size" shall not both be empty. The member reference specifying an EPATH shall consist of either Logical or Data Segments.

The "Member Size" field shall have units of bits. If a "Member Size" field is empty, the defined size of the corresponding "Member Reference" field shall be used. The defined size of a **Param** entry shall be as given in its 6<sup>th</sup> field (size). The defined size of an **Assem** entry shall be as given in its 3<sup>rd</sup> field (size).

The members shall be placed into the data block least significant bit first just as they are in the Assembly object. If a "Member Size" field is smaller than the defined size of the corresponding "Member Reference" field, the least significant bits of the corresponding "Member Reference" field shall be used. If a "Member Size" field is larger than the defined size of the corresponding "Member Reference" field, the entire member shall be followed by zero pads to extend the member to the "Member Size". The data block represented shall be an integer number of bytes. For example, **Assem5** in Figure 7-3.12 shall be 1 byte long and have a default value of 0x21.

**Figure 7-3.12 [Assembly] Section Example**

```
[Params]
Param1 =
0,                      $ first field shall equal 0
6, "20 0F 24 01 30 01", $ path size, path
0x0000,                 $ descriptor
0xD2,                   $ data type : 16-bit WORD
2,                      $ data size in bytes
"Idle state",           $ name
",",                     $ units
"User Manual p48",      $ help string
0, 2, 1,                $ min, max, default data values
0, 0, 0, 0,              $ mult, dev, base, offset scaling not used
0, 0, 0, 0,              $ mult, dev, base, offset link not used
0;                      $ decimal places not used
```

```

Param2 = 0, 6, "20 0F 24 02 30 01",      $ path size, path
      0x0000, 0xD2, 2, "Fault state", "", "User Manual p49",
      0, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0;

[Assembly]
Revision = 2;

Assem5 = "configuration", "20 04 24 05 30 03",1,,,
        4, Param1,
        3, Param2,
        1, ;

```

### 7-3.6.7.2.8 Specifying Bit Fields

The ParamN:b syntax is used to specify which bits from a parameter are to be included in an assembly. The ':b' indicates the bit offset into the parameter, where '0' refers to bit 0. The member reference size indicates the number of relevant bits. It is invalid to specify a size and bit offset that 'overflow' the parameter. For example, the assembly member reference "16, Param1:2" is not valid when the parameter is 16 bits in size. The bits used from the parameter start at the offset specified and continue with the more significant bits for the size specified. This syntax allows disjoint bits within an assembly to be represented with a single parameter.

**Figure 7-3.13 ParamN:b Syntax Example**

```

[Params]
Param1 = 0,                      $ first field shall equal 0
      ,,                      $ path size, path
      0x0002,                  $ descriptor
      0xC6,                    $ data type: Unsigned short integer
      1,                      $ data size in bytes
      "Channel 0 Range Selection Bits", $ name
      "",                      $ units
      "",                      $ help string
      0,3,0,                  $ min, max, default data values
      0,0,0,0,                $ mult, dev, base, offset scaling not used
      0,0,0,0,                $ mult, dev, base, offset link not used
      0;                      $ decimal places
      Enum1 = 0, "Off", 1, "4-20mA", 2, "0-10V", 3, "-10 to +10V"

$ Param2, Param3 & Param4 are similar for Channel 2, Channel 3 & Channel 4

[Assembly]
Assem1 = "Configuration Data", "20 7D 24 00 30 0C",,,,
1,Param1:0,                      $ Channel 0 full range bit
1,Param2:0,                      $ Channel 1 full range bit
1,Param3:0,                      $ Channel 2 full range bit
1,Param4:0,                      $ Channel 3 full range bit
4,,                           $ Not used - must be 0 [fill bits]
1,Param1:1,                      $ Channel 0 selection bit
1,Param2:1,                      $ Channel 1 selection bit
1,Param3:1,                      $ Channel 2 selection bit
1,Param4:1,                      $ Channel 3 selection bit
4,,                           $ Not used - must be 0

```

### 7-3.6.7.3 Variant and VariantExa Entry Keywords

The “**Variant**” and “**VariantExa**” keywords shall be used to describe a dependent member of an assembled data buffer. The “**Variant**” and “**VariantExa**” keywords are similar to a ‘Switch’ statement in a high-level programming language. The ‘Switch’ statement is a multi-way decision that tests whether an expression matches one of a number of constant integer values, and branches accordingly. The “**Variant**” or “**VariantExa**” entry in the [Assembly] section serves a similar purpose. It uses specific bits in an assembly or a parameter to obtain a value called the ‘Switch selector’ that is used to match a number of constant integer values. Once a match is found the corresponding assembly or parameter is used in place of the “**VariantN**” or “**VariantExa**” entry (see Figure 7-3.14 for an example). If multiple matches are found one of the corresponding assemblies or parameters shall be used in place of the “**VariantN**” or “**VariantExa**” entry. The method for selecting which corresponding assembly or parameter requires user intervention. Multiple matches are supported because determination of the device configuration is not available by electronic means.

The entry keyword shall consist of the character array, “**Variant**” or “**VariantExa**”, combined with a decimal number, e.g. “**Variant1**” or “**VariantExa1**”. The same decimal number shall not be used for a “**VariantN**” and a “**VariantExaN**” entry. Commas shall separate all fields, and a semicolon shall indicate the end of the entry. White space or nothing between commas shall be used for optional fields not provided.

“**VariantExaN**” is an extension of the “**VariantN**” keyword. Each “**VariantN**” entry shall contain the formatted fields shown in Table 7-3.27. Each “**VariantExaN**” entry shall contain the formatted fields shown in Table 7-3.28.

**Table 7-3.27 Variant Entry Format**

Field name	Field Number	Data type	Required/Optional
Name	1	Eds_Char_Array	Optional
Help String	2	ASCII Character Array	Optional
Reserved	3	Empty	Optional
Reserved	4	Empty	Optional
Reserved	5	Empty	Optional
Switch selector	6	Param or Assem	Required
First Selection value	7	UINT	Required
First Selection entry	8	Param	Required
Second Selection value	9	UINT	Required
Second Selection entry	10	Param	Required
Subsequent Selection values	11,13,...	UINT	Optional
Subsequent Selection entries	12,14,...	Param	Optional

**Table 7-3.28 VariantExa Entry Format**

Field name	Field Number	Data type	Required/Optional
Name	1	Eds_Char_Array	Optional
Help String	2	ASCII Character Array	Optional
Reserved	3	Reserved	Optional
Reserved	4	Reserved	Optional
Reserved	5	Reserved	Optional
Switch selector	6	Param, Assem or AssemExa	Required
First Selection value	7	UINT	Required
First Selection entry	8	Param, Assem or AssemExa	Required
Second Selection value	9	UINT	Required
Second Selection entry	10	Param, Assem or AssemExa	Required
Subsequent Selection values	11,13,...	UINT	Optional
Subsequent Selection entries	12,14,...	Param, Assem or AssemExa	Optional

#### **7-3.6.7.3.1 Name, Field 1**

The “Name” field shall be a string giving a name to the variant. This optional field may be used by the user interface.

#### **7-3.6.7.3.2 Help String, Field 2**

The “Help String” field shall be a string providing help for the variant. This optional field may be used by the user interface.

#### **7-3.6.7.3.3 Reserved Field 3**

Field 3 shall be reserved for future definition and shall be empty.

#### **7-3.6.7.3.4 Reserved Field 4**

Field 4 shall be reserved for future definition and shall be empty.

#### **7-3.6.7.3.5 Reserved Field 5**

Field 5 shall be reserved for future definition and shall be empty.

### **7-3.6.7.3.6 Switch Selector, Field 6**

The “Switch selector” field shall be an AssemN, ProxyAssemN or ProxiedAssemN entry from the [Assembly] section or a ParamN, ProxyParamN or ProxiedParamN from the [Params] section. A reference to a “symbolic” may use the syntax “symbolic:N1:N2” where N1 specifies a bit offset into the “symbolic” and N2 specifies the number of bits to use. For example, “Assem1:0:4” indicates to use the first four bits of Assem1. The assembly or parameter specified as the “Switch selector” shall exist within the device and therefore shall have a path specified in its definition within the Assembly or Parameter section. The switch selector field shall not specify a portion of an assembly that is a variant.

### **7-3.6.7.3.7 Switch Selection Entry/Value Fields**

The remaining fields shall be paired such that a “Selection value” field is paired with a “Selection entry” field. The “Selection value” field is an unsigned integer and is used in the matching process described below. The “Selection entry” field shall be a ParamN, ProxyParamN, or ProxiedParamN entry from the [Params] section or, for a VariantExaN keyword entry, an AssemN, AssemExaN, ProxyAssemN, ProxyAssemExaN, ProxiedAssemN or ProxiedAssemExaN from the [Assembly] section. The first two “Selection value” / “Selection entry” pairs are required to be specified. Subsequent pairs may also be specified. There is no numeric ordering of the pairs required. Only the matched selection entry specified in a VariantN or VariantExaN shall be visible through the variant, other selection entry(s) in the VariantN or VariantExaN shall be not be visible.

The “Switch selector” field is used to determine a specific “Selection value”. The “Switch selector” value is compared with each of the “Selection value” fields until a match is found. The “Selection entry” paired with the matched “Selection value” is then used as the entry (parameter or assembly) in the assembly. If a match is not found the user interface shall display an error.

**Figure 7-3.14 Variant Keyword Example**

```
[Params]
Param1 =
0,                                $ first field shall equal 0
,,                                 $ path size, path
0x0002,                            $ descriptor
0xC6                               $ data type : Unsigned short integer
1,                                 $ data size in bytes
"Output Channel 0 Configuration",   $ name
",",                               $ units
",",                               $ help string
0,2,0,                            $ min, max, default data values
0,0,0,0,                           $ mult, dev, base, offset scaling not used
0,0,0,0,                           $ mult, dev, base, offset link not used
0;                                $ decimal places not used
Enum1 = 0, "Channel not configured", 1, "4-20mA", 2, "+-10V";

Param2 =
0,                                $ first field shall equal 0
,,                                 $ path size, path
0x0044,                            $ descriptor (scaling and decimal precision)
0xC7,                               $ data type : 16 bit unsigned integer
2,                                 $ data size in bytes
```

```

"Output Channel 0 4-20mA ", $ name
"mA",
"",
0,30840,0,
1,1927,100,7712,
0,0,0,0,
2;                                $ decimal places

Param3 =
0,                                     $ first field shall equal 0
,,                                     $ path size, path
0x0044,                                $ descriptor (scaling and decimal precision)
0xC7,                                   $ data type : 16 bit unsigned integer
2,                                     $ data size in bytes
"Output Channel 0 +-10V ", $ name
"V",
"",
31969,33567,31969,
1,79,100,-32769,
0,0,0,0,
2;                                $ decimal places

Param4 =
0,                                     $ first field shall equal 0
,,                                     $ path size, path
0x0044,                                $ descriptor (scaling and decimal precision)
0xC7,                                   $ data type : 16 bit unsigned integer
2,                                     $ data size in bytes
"Output Channel 0 0-20mA ", $ name
"mA",
"",
0, 31208,0,
1,1560,1000,0,
0,0,0,0,
2;                                $ decimal places

[Assembly]
Assem1 =
"configuration","20 7D 24 00 30 0C", ,,,,
4,Param1;                                $ Output Channel 0 Configuration

Variant1 = "Output Channel 0 Configuration",           $ name
,,,,
Assem1:0:4,                                $ Reserved fields
$ value of Param1 in Assem1 is used to select
$ one of the following parameters
0x01, Param2,                               $ if the above value is 0x01 then use Param2
0x02, Param3,                               $ the user needs to indicate which parameter
$ to user (Param3 or Param4) when the above
$ value is 2
0x02, Param4;                               $ the user needs to indicate which parameter
$ to user (Param3 or Param4) when the above
$ value is 2

Assem2 =
"output" ,,,,
4, Variant1;

```

### 7-3.6.7.4 Examples of Adapter/Rack Assem Data Formatting

The following examples illustrate the concepts of Adapter/Rack Assembly data formatting sections of the EDS file.

#### 7-3.6.7.4.1 Example 1

Example of a fixed slot size (2 bytes per slot), expanded adapter rack Assem, adapter excluded (Assem type = 1). The table below illustrates an example rack with an adapter in a 9-slot rack.

**Table 7-3.29 Rack Module Descriptions for Example 1**

Slot 0	Slot 1	Slot 2	Slot 3	Slot 4	Slot 5	Slot 6	Slot 7	Slot 8
Adapter	16-bit input module	16-bit input module	Empty slot	16-bit output module	16-bit input and output module	16-bit input and output module	16-bit output module	16-bit input module

The input data buffer being described by ProxyAssem1 and ProxiedAssem1 below is:

**Table 7-3.30 Slot Data Descriptions for Example 1**

Slot 1	16 bits of data, format unspecified
Slot 2	4 bits of data described by Param1, 12 bits described by Param2
Slot 3	16 bits of pad
Slot 4	16 bits of pad
Slot 5	4 bits of data described by Param1, 12 bits described by Param2
Slot 6	4 bits of data described by Param1, 12 bits described by Param2
Slot 7	16 bits of pad
Slot 8	4 bits of data described by Param1, 12 bits described by Param2

The following adapter and module EDS snippets could be used to describe the adapter rack input Assem for the example rack:

**Figure 7-3.15 EDS File Entries (partial) for Example 1**

```

Adapter EDS:
ProxyAssem1 =
    "Module Input Data",      $ Assem name
    "20 04 24 SLOT 30 0A",   $ no path
    2,                      $ 2 bytes of input per slot
    0b0000000000000010,     $ fixed slot size, expanded adapter rack
                           $ Assem, adapter excluded
    , ,                     $ reserved, reserved
    ModuleMemberList;        $ the module EDS specifies the format of
                           $ the module input data

Module in slot 1 EDS:
ProxiedAssem1 = , , , , , $ Assem name, path, size, descriptor,
                           $ reserved, reserved
    16, ;                  $ 16 bits of input data, format unspecified

Module in slot 2 EDS:
ProxiedAssem1 = , , , , , $ Assem name, path, size, descriptor,
                           
```

<pre> \$ reserved, reserved 4, Param1,           \$ 4 bits of input data 12, Param2;          \$ 12 bits of input data </pre>
<p>Module in slot 4 EDS:</p> <pre> ProxiedAssem1 = ,,,,,,      \$ Assem name, path, size, descriptor,                            \$ reserved, reserved ;                            \$ no input data </pre>
<p>Module in slot 5 EDS:</p> <pre> ProxiedAssem1 = ,,,,,,      \$ Assem name, path, size, descriptor,                            \$ reserved, reserved 4, Param1,               \$ 4 bits of input data 12, Param2;              \$ 12 bits of input data </pre>
<p>Module in slot 6 EDS:</p> <pre> ProxiedAssem1 = ,,,,,,      \$ Assem name, path, size, descriptor,                            \$ reserved, reserved 4, Param1,               \$ 4 bits of input data 12, Param2;              \$ 12 bits of input data </pre>
<p>Module in slot 7 EDS:</p> <pre> ProxiedAssem1 = ,,,,,,      \$ Assem name, path, size, descriptor,                            \$ reserved, reserved ;                            \$ no input data </pre>
<p>Module in slot 8 EDS:</p> <pre> ProxiedAssem1 = ,,,,,,      \$ Assem name, path, size, descriptor,                            \$ reserved, reserved 4, Param1,               \$ 4 bits of input data 12, Param2;              \$ 12 bits of input data </pre>

#### 7-3.6.7.4.2 Example 2

Example of a fixed slot size (2 bytes per slot), compressed adapter rack Assem, adapter excluded (Assem type = 2). The table below illustrates an example rack with an adapter in a 9-slot rack.

**Table 7-3.31 Rack Module Descriptions for Example 2**

Slot 0	Slot 1	Slot 2	Slot 3	Slot 4	Slot 5	Slot 6	Slot 7	Slot 8
Adapter	16-bit input module	16-bit input module	Empty slot	16-bit output module	16-bit input and output module	16-bit input and output module	16-bit output module	16-bit input module

The input data buffer being described by ProxyAssem1 and ProxiedAssem1 below is:

**Table 7-3.32 Slot Data Descriptions for Example 2**

Slot 1	16 bits of data, format unspecified
Slot 2	4 bits of data described by Param1, 12 bits described by Param2
Slot 5	4 bits of data described by Param1, 12 bits described by Param2
Slot 6	4 bits of data described by Param1, 12 bits described by Param2
Slot 8	4 bits of data described by Param1, 12 bits described by Param2

The following adapter and module EDS snippets could be used to describe the adapter rack input Assem for the example rack:

**Figure 7-3.16 EDS File Entries (partial) for Example 2**

Adapter EDS:

```
ProxyAssem1 =
    "Module Input Data",      $ Assem name
    "20 04 24 SLOT 30 0A",   $ no path
    2,                        $ 2 bytes of input per slot
    0b0000000000000100,      $ fixed slot size, compressed adapter
    ..                        $ rack Assem, adapter excluded
    ModuleMemberList;         $ reserved, reserved
                            $ the module EDS specifies the format of
                            $ the module input data
```

Module in slot 1 EDS:

```
ProxiedAssem1 = .....      $ Assem name, path, size, descriptor,
                            $ reserved, reserved
    16,;                     $ 16 bits of input data, format unspecified
```

Module in slot 2 EDS:

```
ProxiedAssem1 = .....      $ Assem name, path, size, descriptor,
                            $ reserved, reserved
    4, Param1,              $ 4 bits of input data
    12, Param2;             $ 12 bits of input data
```

Module in slot 5 EDS:

```
ProxiedAssem1 = .....      $ Assem name, path, size, descriptor,
                            $ reserved, reserved
    4, Param1,              $ 4 bits of input data
    12, Param2;             $ 12 bits of input data
```

Module in slot 6 EDS:

```
ProxiedAssem1 = .....      $ Assem name, path, size, descriptor,
                            $ reserved, reserved
    4, Param1,              $ 4 bits of input data
    12, Param2;             $ 12 bits of input data
```

Module in slot 8 EDS:

```
ProxiedAssem1 = .....      $ Assem name, path, size, descriptor,
                            $ reserved, reserved
    4, Param1,              $ 4 bits of input data
    12, Param2;             $ 12 bits of input data
```

### **7-3.6.7.4.3 Example 3**

Example of a variable sized, compressed adapter rack Assem, adapter excluded (Assem type = 3). The table below illustrates an example adapter rack with an adapter in a 9-slot rack.

**Table 7-3.33 Rack Module Descriptions for Example 3**

<b>Slot 0</b>	<b>Slot 1</b>	<b>Slot 2</b>	<b>Slot 3</b>	<b>Slot 4</b>	<b>Slot 5</b>	<b>Slot 6</b>	<b>Slot 7</b>	<b>Slot 8</b>
Adapter	16-byte input module	16-byte input module	Empty slot	8-byte output module	16-byte input and 4-byte output module	16-byte input and 4 byte output module	8-byte output module	2-byte input module

The input data buffer described by ProxyAssem1 and the ProxiedAssem1's is:

**Table 7-3.34 Slot Data Descriptions for Example 3**

Slot 1	128 bits of data, format unspecified
Slot 2	4 bits of data described by Param1, 12 bits described by Param2
Slot 5	128 bits of data, format unspecified
Slot 6	128 bits of data, format unspecified
Slot 8	4 bits of data described by Param1, 12 bits described by Param2

The following adapter and module EDS snippets could be used to describe the adapter rack input Assem for the example rack:

**Figure 7-3.17 EDS File Entries (partial) for Example 3**

Adapter EDS:

```
ProxyAssem1 =
    "Module Input Data",      $ Assem name
    "20 04 24 SLOT 30 0A",   $ no path
    ,                         $ size not specified, use member list size
    0b00000000000000110,     $ variable sized, compressed adapter rack
    $ Assem, adapter excluded
    ..
    $ reserved, reserved
    ModuleMemberList;         $ the module EDS specifies the format of
    $ the module input data
```

Module in slot 1 EDS:

```
ProxiedAssem1 = ,,,,,,      $ Assem name, path, size, descriptor,
                           $ reserved, reserved
                           128,;                   $ 128 bits of input data, format unspecified
```

Module in slot 2 EDS:

```
ProxiedAssem1 = ,,,,,,      $ Assem name, path, size, descriptor,
                           $ reserved, reserved
                           4, Param1,             $ 4 bits of input data
                           12, Param2;            $ 12 bits of input data
```

Module in slot 5 EDS:

```
ProxiedAssem1 = ,,,,,,      $ Assem name, path, size, descriptor,
                           $ reserved, reserved
                           128;                   $ 128 bits of input data, format unspecified
```

Module in slot 6 EDS:

```
ProxiedAssem1 = ,,,,,,      $ Assem name, path, size, descriptor,
                           $ reserved, reserved
                           128;                   $ 128 bits of input data, format unspecified
```

Module in slot 8 EDS:

```
ProxiedAssem1 = ,,,,,,      $ Assem name, path, size, descriptor,
                           $ reserved, reserved
                           4, Param1,             $ 4 bits of input data
                           12, Param2;            $ 12 bits of input data
```

### 7-3.6.8 Complete Electronic Data Sheet Example

An Electronic Data Sheet example is given below which is a compilation of the section examples given thus far in this chapter.

**Figure 7-3.18 Electronic Data Sheet Example**

```
[File]
  DescText = "Smart Widget EDS File";
  CreateDate = 04-03-1994;           $ created
  CreateTime = 17:51:44;
  ModDate = 04-06-1994;           $ last changed
  ModTime = 22:07:30;
  Revision = 2.1;                  $ revision of EDS

[Device]
  VendCode = 65535;
  VendName = "Widget-Works, Inc.";
  ProdType = 0;
  ProdTypeStr = "Generic";
  ProdCode = 42;
  MajRev = 1;
  MinRev = 1;
  ProdName = "Smart-Widget";
  Catalog = "1499-DVG";

[ParamClass]
  MaxInst = 3;
  Descriptor = 0x0E;
  CfgAssembly = 3;

[Params]
  Param1 = 0, 1, "20 02", 0x0014, 0xC6, 1, "Preset", "V", "User Manual p33",
           0, 5, 1, 1, 1, 1, 0, 0, 0, 0, 0, 2;
  Param2 =                               $ parameter instance
           0,                      $ First field shall equal 0
           1, "20 04",             $ path size, path
           0x0014,                 $ descriptor - in hex format
           0xC6,                   $ data type: Unsigned 8-Bit Integer (USINT)
           1,                      $ data size
           "Trigger",              $ name
           "Hz",                   $ units
           "User Manual p49",      $ help string
           0, 2, 0,                $ min, max, default data values
           1, 1, 1, 0,              $ mult, div, base, offset scaling
           0, 0, 0, 0,              $ mult, div, base, offset links not used
           2;                      $ decimal places

[Groups]
  Group1 = "Setup", 2, 1, 2;        $ group 1
  Group2 = "Monitor", 2, 2, 3;       $ group 2
  Group3 = "Maintenance", 2, 1, 3;   $ group 3
```

### 7-3.6.9 Connection Manager Section

The Connection Manager section defines the set of target I/O connections supported by this device. These target I/O connections may exist as static connections predefined in the device or they may be dynamic I/O target connections which need to be configured in the device. Devices that support the Connection Configuration object may support the ability to dynamically create target I/O connections.

The Connection Manager section begins with the keyword [Connection Manager] and shall contain information concerning the number of types of application connections a device supports. Each entry keyword shall be one of the following character arrays, “**Connection**”, “**ProxyConnect**”, “**ProxiedConnect**”, combined with a number (decimal), for example, “**Connection1**”, “**ProxyConnect1**”, or “**ProxiedConnect1**”. The decimal numbers shall start at 1 and increment for each additional “Connection” entry. The decimal number shall not be required to start at 1 or increment for each additional “**ProxyConnect**” or “**ProxiedConnect**” entry. Commas shall separate all fields, and a semicolon shall indicate the end of the entry. A white space or nothing between commas shall be used for optional fields not provided. The “**ProxyConnect**” and “**ProxiedConnect**” keywords are defined further in Chapter 7-3.7, Modular EDS File Requirements.

Each entry shall contain the formatted fields shown in Table 7-3.35.

**Table 7-3.35 Fields in the Connection Manager Section**

Field name	Field number	Data type	Required/Optional
Trigger and transport	1	DWORD	Required
Connection parameters	2	DWORD	Required
O->T RPI	3	UDINT or Param	Optional
O->T size	4	UINT or Param	Conditional
O->T format	5	Param or Assem or AssemExa	Conditional
T->O RPI	6	UDINT or Param	Optional
T->O size	7	UINT or Param	Conditional
T->O format	8	Param or Assem or AssemExa	Conditional
config #1 size	9	UINT or Param	Optional
config #1 format	10	Param or Assem or AssemExa	Optional
config #2 size	11	UINT or Param	Optional
config #2 format	12	Param or Assem or AssemExa	Optional
Connection name string	13	Eds_Char_Array	Required
Help string	14	Eds_Char_Array	Required
Path	15	Eds_Char_Array	Required
Safety ASYNC	16	See CIP Safety Spec (Volume 5, Chapter 5)	Conditional <sup>1</sup>
Safety Max Consumer Number	17		Conditional <sup>1</sup>

1 - These fields are not allowed if the connection is not a safety connection. If the connection is a safety connection, see Volume 5.

### **7-3.6.9.1 Trigger and transport mask**

The bit assignments for the trigger and transport mask shall be as shown in Table 7-3.36 A bit shall be set to a 1 (on) for each trigger mode the connection supports. All other bits shall be set to a 0 (off). For the client/server bit: 0=client, 1=server. Only one of the transport types shall be set to a 1 (on).

**Table 7-3.36 Trigger and transport mask bit assignments**

<b>Bit</b>	<b>Bit definition</b>
0	class 0: null
1	class 1: duplicate detect
2	class 2: acknowledged
3	class 3: verified
4	class 4: non-blocking
5	class 5: non-blocking, fragmenting
6	class 6: multicast, fragmenting
7-15	class: reserved
16	trigger: cyclic
17	trigger: change of state
18	trigger: application
19-23	trigger: reserved
24	transport type : listen-only
25	transport type : input-only
26	transport type : exclusive-owner
27	transport type : redundant-owner
28-30	reserved
31	client = 0 / server = 1

### **7-3.6.9.2 Connection parameters**

The bit assignments for the connection type and priority mask shall be as shown in Table 7-3.37. A bit shall be set to a 1 (on) for each connection type and priority the connection supports. All other bits shall be set to a 0 (off).

**Table 7-3.37 Connection Parameters Bit Assignments**

<b>Bit</b>	<b>Bit definition</b>
0	O->T fixed size supported
1	O->T variable size supported
2	T->O fixed size supported
3	T->O variable size supported

Bit	Bit definition
4 – 5	Obsolete, this capability now exists in AssemN descriptor bits 1-3 (Assem Type). O->T number of bytes per slot in the O->T real time data packet for adapter rack connections. 0 = 1 byte 1 = 2 bytes 2 = 4 bytes 3 = 8 bytes
6 – 7	Obsolete, this capability now exists in AssemN descriptor bits 1-3 (Assem Type). T->O number of bytes per slot in the T->O real time data packet for adapter rack connections. 0 = 1 bytes 1 = 2 bytes 2 = 4 bytes 3 = 8 bytes
8 – 10	O->T Real time transfer format. 0 = connection is pure data and is modeless. 1 = Use zero data length packet to indicate idle mode. 2 = reserved 3 = heartbeat 4 = 32-bit run/idle header 5 thru 7 are reserved
11	reserved
12 – 14	T->O Real time transfer format 0 = connection is pure data and is modeless. 1 = Use zero data length packet to indicate idle mode 2 = reserved 3 = heartbeat. 4 = 32-bit run/idle header 5 thru 7 are reserved
15	reserved
16	O->T connection type: NULL
17	O->T connection type: MULTICAST
18	O->T connection type: POINT2POINT
19	O->T connection type: reserved
20	T->O connection type: NULL
21	T->O connection type: MULTICAST
22	T->O connection type: POINT2POINT
23	T->O connection type: reserved
24	O->T priority: LOW
25	O->T priority: HIGH
26	O->T priority: SCHEDULED
27	O->T priority: reserved
28	T->O priority: LOW
29	T->O priority: HIGH
30	T->O priority: SCHEDULED
31	T->O priority: reserved

#### **7-3.6.9.3 O->T RPI**

The O->T RPI shall be the number of microseconds of the requested packet interval. The O->T RPI shall be a UDINT or a Param entry from the [Params] section that evaluates to a UDINT. If this field is empty, no constraints are placed on the O->T RPI.

#### **7-3.6.9.4 O->T size**

The O->T size shall be the number of bytes sent in the O->T packet excluding the conditional sequence count (the sequence count does not exist for class 0 connections) and conditional 32-bit real time header (the real time transfer format is specified by bits 8-10 of the connection parameters field) sizes. The O->T size shall be a UINT or a Param entry that evaluates to a UINT. If this field is empty, the defined size of the O->T format shall be used.

#### **7-3.6.9.5 O->T format**

The O->T format entry shall define the structure of the consumer buffer for this connection. Valid format descriptors shall be identifiers within the EDS file including

- a Param entry from the [Params] section;
- an Assem/AssemExa entry from the [Assembly] section.

This field may be empty indicating that the consuming format is not specified. This field shall not be empty if the O->T size field is empty. The O->T format shall not include the 32-bit real time header if it is present.

When the O->T format field is not empty and the O->T size field is not empty, if the O->T size field is smaller than the defined size of the O->T format, the least significant bytes of the O->T format fields shall be used. When the O->T format field is not empty and the O->T size field is not empty, if the O->T size field is larger than the defined size of the O->T format, the entire O->T format field shall be followed by zero pads to extend the O->T format to the “O->T Size”.

#### **7-3.6.9.6 T->O RPI**

The T->O RPI shall be the number of microseconds of the requested packet interval. The T->O RPI shall be a UDINT or a Param entry from the [Params] section that evaluates to a UDINT. If this field is empty, no constraints are placed on the T->O RPI.

#### **7-3.6.9.7 T->O size**

The T->O size shall be the number of bytes sent in the T->O packet excluding conditional sequence count (the sequence count does not exist for class 0 connections) and conditional 32-bit real time header (the real time transfer format is specified by bits 12-14 of the connection parameters field) sizes. The T->O size shall be a UINT or a Param entry that evaluates to a UINT. If this field is empty, the defined size of the T->O format shall be used.

#### **7-3.6.9.8 T->O format**

The T->O format shall define the structure of the producer buffer for this connection. Valid format descriptors shall be identifiers within the EDS file including

- a Param entry from the [Params] section;
- an Assem/AssemExa entry from the [Assembly] section.

This field may be empty indicating that the producing format is not specified. This field shall not be empty if the T->O size field is empty. The format shall include the 32-bit real time header if it is present.

When the T->O format field is not empty and the T->O size field is not empty, if the T->O size field is smaller than the defined size of the T->O format, the least significant bytes of the T->O format fields shall be used. When the T->O format field is not empty and the T->O size field is not empty, if the T->O size field is larger than the defined size of the T->O format, the entire T->O format field shall be followed by zero pads to extend the T->O format to the “T->O Size”.

#### **7-3.6.9.9 Configuration**

The config #1 size and config #2 size shall specify the size of the optional data segment that is appended to the path in the Forward\_Open. The data segment shall be the concatenation of the two buffers described by the config #1 format and config #2 format. The sizes shall be the number of bytes and shall be a UINT or a Param entry from the [Params] section that evaluates to a UINT. If one of the config size fields is empty, the natural size of the corresponding config format field shall be used.

Valid config format fields shall be identifiers within the EDS file including

- a Param entry from the [Params] section;
- an Assem/AssemExa entry from the [Assembly] section.

The config format fields may be empty indicating that the config format is not specified. If both the config size and config format fields are empty, no data segment shall be appended to the path of the Forward\_Open.

#### **7-3.6.9.10 Connection Name String**

A tool may display the connection name string (character array). The connection name string shall be unique among all Connection entries within the EDS.

#### **7-3.6.9.11 Help string**

A tool may display the textual help character array. If no help string is to be provided a “null” string shall be used where a null string is defined as two double quotations: “” with no characters between the quotation marks.

#### **7-3.6.9.12 Path**

A path referencing the target object. The path shall be entered as a CIP Path (character array).

**Figure 7-3.19 [Connection Manager] Section Example**

```
[Params]
Param1 =
0, , ,
0x0004,
0xC6, 1,
"Read",
", ,",
64, 95, 64,
1, 1, 1, -63,
0, 0, 0, 0, 0;
$ specifies read buffer
$ no path means not directly accessible
$ descriptor : support scaling
$ USINT, 1 byte
$ name
$ units & help string
$ min, max, default data values
$ mult, div, base, offset scaling
$ mult, div, base, offset link & decimal (not used)

Param2 =
0, , ,
0x0004,
0xC6, 1,
"Write",
", ,",
160, 191, 160,
1, 1, 1, -159,
0, 0, 0, 0, 0;
$ specifies write buffer
$ no path means not directly accessible
$ descriptor : support scaling
$ USINT, 1 byte
$ name
$ units & help string
$ min, max, default data values
$ mult, div, base, offset scaling
$ mult, div, base, offset link & decimal $(not used)

[Connection Manager]
Connection1 =
0x04010002,      $ trigger & transport
                  $ class 1, cyclic, exclusive-owner
0x44244401,      $ point/multicast & priority & real time format
                  $ fixed, 32-bit headers, scheduled,
                  $ O->T point-to-point, T->O multicast
, 16, ,
, 12, ,
, ,
, ,
"read/write",    $ connection name
", ,",
"20 04 24 01 2C [Param2] 2C [Param1]";
```

### 7-3.6.10 Port Section

The Port section begins with the keyword [Port] and shall describe the CIP ports available within a device. Every CIP shall have a corresponding entry in this section. The entry keyword for all ports shall consist of the character array “Port”, combined with a decimal number corresponding to an instance of the port object. For example, Port1 is instance 1 of the Port Object. A white space or nothing between commas shall be used for optional fields not provided.

**Table 7-3.38 Fields in the Port Section**

Field Name	Field number	Data type	Required/Optional
Port Type Name	1	Field Keyword	Required
Port Name	2	Eds_Char_Array	Conditional <sup>1</sup>
Port Object	3	Eds_Char_Array	Optional
Port Number	4	UINT	Required
Reserved	5, 6	Shall be empty	Not Used
Port Specific	7, 8, ...	Port Specific	Port Specific

1 This field is required if the CIP port shares a physical network port with another CIP port.

The first field, called “Port Type Name”, shall be one of

- ControlNet
- ControlNet\_Redundant
- TCP<sup>1</sup>
- DeviceNet
- CompoNet
- ModbusSL
- ModbusTCP
- A vendor-specific value beginning with the device’s Vendor ID and an underscore character (‘65535\_’).

**Figure 7-3.20 [Port] Section Example**

```
[Port]
  Port1 = DeviceNet,
    "Port A",           $ name of port
    "20 03 24 01",     $ instance one of the DeviceNet object
    2;                 $ port number 2

  Port2 = 65535_Chassis,
    "Chassis",          $ name of port
    "20 9A 24 01",     $ vendor specific back-plane object
    1;                 $ port number 1
```

The “Port Name” field shall be a string giving a name to the **physical network** port, and may be used by a user interface. If [multiple Port entry keywords contain the same Port Name field value, then those CIP ports share the same physical network port. See the Port Name attribute \(Attribute ID 4\) of the Port Object in Chapter 3 for the definition of this field.](#) The “Port Object” field shall be a path that identifies the object associated with the port.

The port number 1 shall correspond to the back plane “port”. Devices with a back plane that cannot send CIP messages shall not have a port number 1.

<sup>1</sup> This shall indicate an EtherNet/IP capable TCP port

### **7-3.6.11 Capacity Section**

The Capacity section shall contain information relating to a device's communications capacity. Capacity section keywords are primarily oriented towards specifying a device's I/O communications capacity. The Capacity section also contains parameters not specifically related to I/O.

The entries in the Capacity section are intended to allow a software tool to walk through a set of configured connections for a system and determine whether each of the devices will be able to handle the configured load.

The following table shows the keywords defined for the Capacity section:

**Table 7-3.39 Capacity Section keywords**

Entry Name	Entry Keyword	Number of Fields	Data Type	Required/Optional
Traffic Spec	TSpecN	3	n/a	Optional
Connection overhead	ConnOverhead	1	REAL	Optional
Maximum CIP connections	MaxCIPConnections	1	UINT	Conditional
Maximum I/O connections	MaxIOConnections	1	UINT	Conditional
Maximum explicit messaging connections	MaxMsgConnections	1	UINT	Conditional
Maximum I/O producers	MaxIOProducers	1	UINT	Conditional
Maximum I/O consumers	MaxIConsumers	1	UINT	Conditional
Maximum I/O producer plus consumers	MaxIOProduceConsume	1	UINT	Conditional
Maximum I/O multicast producers	MaxIOMcastProducers	1	UINT	Conditional
Maximum I/O multicast consumers	MaxIOMcastConsumers	1	UINT	Conditional
Maximum consumers per multicast connection	MaxConsumersPerMcast	1	UINT	Conditional

### **7-3.6.11.1 Traffic Spec**

The Traffic Spec keyword shall be used to indicate a device's maximum implicit communications packet rate (packets per second) across the range of supported CIP connection sizes. An EDS file will typically have a set of Traffic Spec keywords, TSpec1 through TSpec $N_{max}$ , to describe the device's maximum implicit packet rate across different connection sizes.

The following table shows the format of the Traffic Spec keyword:

**Table 7-3.40 Traffic Spec Keyword Format**

Field Name	Field Number	Data Type	Required/Optional
TxRx	1	ASCII Character Array	Required
ConnSize	2	UINT	Required
PacketsPerSecond	3	UINT	Required

- **TxRx** shall have one of the following values: “Tx”, “Rx”, or “TxRx”. Tx shall indicate that the Traffic Spec applies to packets transmitted (produced) by the device. “Rx” shall indicate that the Traffic Spec applies to packets received (consumed) by the device. “TxRx” shall indicate that the Traffic Spec applies to both packets transmitted and received. If any of the Traffic Spec entries in a Capacity section use “TxRx” then ALL entries in the section shall use “TxRx”.
- **ConnSize** shall be the CIP connection size, in bytes. That is, the ConnSize corresponds to the O->T and T->O connection size in the Forward Open service, or to the Produced\_connection\_size and Consumed\_connection\_size instance attributes of the Connection Object.
- **PacketsPerSecond** shall be the maximum number of implicit packets per second the device can process at the given ConnSize.

A Traffic Spec is a point on the connection size vs. packets per second curve. Between any two adjacent points, the curve is assumed to be linear. Devices can specify as few or as many different traffic specs as needed to characterize their capacity at different connection sizes.

As an example, a device might have the following Traffic Specs:

- TSpec1 = TxRx, 2, 5000;
- TSpec2 = TxRx, 128, 4700;
- TSpec3 = TxRx, 256, 4000;
- TSpec4 = TxRx, 508, 2800;

In the example above, at a connection of size 2, the device can support 5000 packets per second. At a connection size of 128, the device can support 4700 packets per second. Linear interpolation is used to determine the capacity (packets per second) between those two sizes.

TSpecN entries shall be numbered consecutively, starting with N = 1. TSpecN entries shall be listed in order of increasing ConnSize. The first TSpec entry (TSpec1) shall use the smallest ConnSize supported by the device. The last TSpec entry shall use the largest ConnSize supported by the device.

Traffic Specs apply to a device’s implicit communications capacity only. For EtherNet/IP and ControlNet devices, Traffic Specs shall apply to implicit connections that specify transport class 0 or transport class 1. For DeviceNet devices, Traffic Specs shall apply to implicit connections regardless of the transport class.

### **7-3.6.11.2 Generating TSpec Values**

Each TSpec shows the device’s maximum implicit packets per second at a given connection size. TSpecs should be generated over the range of connection sizes supported by the device. At each chosen connection size, the maximum packets per second should be determined using the smallest (fastest) RPI over the fewest number of connections possible. The recommended procedure to generate a Traffic Spec is outlined below:

- Establish an I/O connection to the device (or, from the device if it is an originator). Note: packets sent on the connection shall be a fixed size.
- Start with smallest RPI supported by the device.
- Measure the sustained packet rate (using a network analyzer, device diagnostics, or other means).

- Determine whether packets are being lost (using a network analyzer, device diagnostics, observing connection loss, or by other means).
- If there is packet loss, then increase the RPI (decrease packet rate) until no packets are being lost.
- If no packet loss, and if the RPI cannot be made any faster, then add connections until the device experiences packet loss or runs out of connections (RPIs for subsequent connections may need to vary).
- PacketsPerSecond for the TSpec shall be the measured packet rate. If the T->O and O->T (produced and consumed) connection sizes are not equal, then additional calculations must be done to generate the PacketsPerSecond number (see below).

Vendors should determine whether a device has the same capacity for Tx and Rx packets. If the device has a significant difference in Tx and Rx capacities, then it is desirable, though not required, to generate different Tx and Rx TSpecs. There are several options for generating separate Tx and Rx TSpecs:

1. using a null connection type in one direction
2. measuring Rx packet rates while holding the Tx packet size and rate constant (and vice versa for Tx packets)

Generating TSpec values can be difficult in that devices and the application software may support a limited number of connection sizes, and the connection sizes may be different in the T->O and O->T (produced and consumed) directions. For example, consider a device where the T->O connection size is 256, the O->T connection size is 2, and the maximum packet rate is measured at 4000 packets per second (2000 in each direction). Since different packet sizes are used, it is not possible to conclusively determine the maximum packet rates for the 2-byte and 256-byte sizes. We assume that at 2 bytes the device can process at least 4000 packets per second. At 256 bytes, we assume the device can process somewhere between 2000 and 4000 packets per second. In this situation, the recommended approach is to estimate the maximum packet rates at each of the two connection sizes, and then validate experimentally by setting up a number of connections and measuring packet rates or device CPU utilization if available.

It is acceptable for vendors to generate TSpecs in accordance with the device's limitations. For example, devices may support limited connection sizes and RPIs. If a device supports only a single fixed connection size then the device would have only one TSpec. If the device supports multiple connection sizes, then there shall be at least one TSpec at the minimum size, and one at the maximum size. Between the minimum and maximum, vendors should provide enough TSpecs to adequately characterize the I/O capacity of their devices.

After generating TSpec values, vendors should validate the TSpecs by setting up actual application connections over the range of supported connection sizes, RPI's, and number of connections to ensure that the device supports what the TSpecs state.

### **7-3.6.11.3 Connection Overhead**

The number of connections can also have an affect on a device's I/O capacity, independent of connection size and RPI. For example, due to connection management overhead, a device may be able to process 4000 packets with 2 connections, but only 3000 packets with 128 connections (given the same packet size).

To account for connection overhead, the ConnOverhead keyword shall indicate a processing overhead to add per connection. ConnOverhead shall be expressed as a decimal number, representing the percentage of processing overhead for each connection (e.g., a value of .002 represents .2% overhead per connection). The value of ConnOverhead shall be less than 1.

Refer to the section on 7-3.6.11.6, Calculating Implicit Communications Usage for a description of how ConnOverhead is used.

The following shows an example of a ConnOverhead entry:

ConnOverhead = .002;

In the above example, for each connection there is a processing overhead of .002 (or .2%).

Devices are not required to include a ConnOverhead entry. Devices that do not incur additional overhead related to connection processing would typically not include a ConnOverhead entry.

#### **7-3.6.11.4 Determining ConnOverhead Values**

Values for the “ConnOverhead” parameters shall be determined by one of the following methods:

1. At a given packet rate, measure the device’s CPU utilization (if available and accurate) at 1 connection and at the maximum number of connections supported by the device. ConnOverhead is the percent difference (expressed as a decimal number) in CPU utilization at 1 connection and at the maximum number of connections.
2. At a given connection size, measure the maximum packet rate at 1 connection (or at the smallest number of connections needed to generate max packet rate), and at the maximum connections supported by the device. ConnOverhead is the percent difference (expressed as a decimal number) in the maximum packet rate at 1 connection and at the maximum number of connections.

If possible, devices should determine their Traffic Specs using the smallest number of connections (ideally 1), and then determine their ConnOverhead using the methods outlined above. Otherwise, if the Traffic Specs are generated using a significant number of connections (relative to the maximum connections supported), then the connection-processing overhead will be accounted for in the Traffic Spec.

Some devices may have difficulty identifying connection overhead independent of their Traffic Specs. For example, a device that supports RPIS no faster than 50 ms may require a significant number of connections in order to generate a Traffic Spec. In this case, connection-processing overhead is already accounted for in the Traffic Spec. If the device also includes a ConnOverhead entry, the connection overhead would be included a second time.

If a device requires a significant number of connections (relative to the maximum number of connections supported) to generate its Traffic Specs, then the device should generally not include a ConnOverhead entry.

### **7-3.6.11.5 Capacity Assumptions**

It is important that EDS capacity parameters be consistently interpreted and reported across different products. Values for capacity parameters – Traffic Spec rates, Connection Overhead, etc. – shall be assumed to indicate the device capacity that can be sustained under normal operating conditions in the absence of other (non-I/O) traffic, including:

- other CIP traffic (e.g., Class 3)
- non-CIP traffic (e.g., HTTP for EtherNet/IP devices, etc.)
- rejected multicast packets (due to limitations of Ethernet multicast filtering)

Future specification enhancements could address the effects of non-I/O traffic on I/O capacity.

### **7-3.6.11.6 Calculating Implicit Communications Usage**

There are no requirements for particular behavior of a software tool that uses the capacity parameters. This section suggests an algorithm that could be used by a software tool to calculate and report device capacity.

A software tool would typically use the EDS capacity parameters in order to determine an application's projected communications usage. The tool would calculate, for each device in the application, the percentage of implicit communications capacity required by the configured connections. The tool could then report the percent utilization for each device, and whether any device has exceeded its capacity.

To calculate projected implicit communications usage, the software tool might do the following:

For each connection originator

For each configured implicit connection

Calculate the O->T (consumed) packet rate based on O->T RPI

Find the TSpec corresponding to the O->T size. If the connection size is variable, assume all packets are the largest size. Use linear interpolation if between TSpecs.

Determine the percentage of packet capacity used by O->T traffic and add to the sum of connection percentages for both originator and target

Calculate the T->O (produced) packet rate based on T->O RPI

Find the TSpec corresponding to the T->O size. If the connection size is variable, assume all packets are the largest size. Use linear interpolation if between TSpecs.

Determine the percentage of packet capacity used by T->O traffic and add to sum of connection percentages for both originator and target

For each connection originator and for each target

Multiply the ConnOverhead times the number of connections and add to the sum of connection percentages.

### **7-3.6.11.7 Connection Capacity Keywords**

In addition to implicit communications capacity, it is useful for a device to provide information on the number of CIP connections supported, for both I/O and explicit communications. Connection capacity keywords are an important factor in allowing the user to determine whether the system as configured can successfully operate.

Characterizing a device's connection capacity is complicated by several factors:

- Depending upon implementation, multicast connections may share connection resources if they produce or consume the same data.
- CIP implementations may use different models for counting and limiting their connection resources

In order to account for different implementations, a number of connection capacity keywords are defined (refer to Table 7-3.41). As documented in Chapter 3, the term "connection" refers to the communications entity created using the Create service of the Connection Object, or the Forward\_Open service of the Connection Manager Object.

The connection capacity keywords are further described below:

- **Maximum CIP connections (MaxCIPConnections)** – the maximum number CIP connections supported, including both I/O and explicit connections.
- **Maximum I/O connections (MaxIOConnections)** – the maximum number of I/O connections.
- **Maximum explicit connections (MaxMsgConnections)** – the maximum number of explicit message connections.
- **Maximum I/O producers (MaxIOProducers)** – the maximum number of I/O link producers.
- **Maximum I/O consumers (MaxIOConsumers)** – the maximum number of I/O link consumers.
- **Maximum I/O producers plus consumers (MaxIOProduceConsume)** – the maximum total of I/O link producers and consumers combined.
- **Maximum I/O multicast producers (MaxIOMcastProducers)** – the maximum number of multicast link producers.
- **Maximum I/O multicast consumers (MaxIOMcastConsumers)** – the maximum number of multicast link consumers.
- **Maximum consumers per multicast connection (MaxConsumersPerMcast)** – the maximum number of consumers allowed to join any one multicast connection.

The connection capacity keywords are conditional. The keywords may be used in particular combinations to model different approaches to counting and limiting connection resources. The allowable combinations of connection capacity keywords are show in the following table:

**Table 7-3.41 Allowable Combinations of Capacity Keywords**

Allowed Combination of Connection Capacity Keywords	Description of Connection Counting Model	Additional Capacity Keywords Allowed (Optional)
MaxCIPConnections	The device counts CIP connections, with a common pool for both I/O and explicit message connections	MaxConsumersPerMcast
MaxIOConnections MaxMsgConnections	The device counts CIP connections, with separate limits on I/O and explicit messaging connections.	MaxConsumersPerMcast
MaxIOProduceConsume MaxMsgConnections	The device counts I/O and explicit message connections differently. For I/O connections the device counts producers and consumers. Each connection requires zero or one producer and zero or one consumer. Producers and consumers share a common pool. Multicast connections that request the same data are able to share producer and consumer resources.  For explicit messaging connections the device counts connections.	MaxIOMcastProducers MaxIOMcastConsumers MaxConsumersPerMcast
MaxIOProducers MaxIOConsumers MaxMsgConnections	Same as the above model, except I/O producers and consumers are maintained in separate pools.	MaxIOMcastProducers MaxIOMcastConsumers MaxConsumersPerMcast

The connection capacity keywords are typically a result of limits built into a product's firmware or software. No measurement is generally needed to determine these limits, however it is expected that a product shall support the limits as stated in its EDS file.

### 7-3.6.11.8 Considerations for Multiport Devices

Devices may have multiple ports, of the same type or different types. The keywords in the [Capacity] section shall apply device-wide, across all ports.

**Figure 7-3.21 Sample Electronic Data Sheet illustrating the Capacity Section**

```
[File]
...
[Device]
...
[Capacity]
    TSpec1 = TxRx, 2, 5000;      $ packets per sec @ 2 bytes
    TSpec2 = TxRx, 128, 4700;    $ packets per sec @ 128 bytes
    TSpec3 = TxRx, 256, 4200;    $ packets per sec @ 256 bytes
    TSpec4 = TxRx, 508, 3400;    $ packets per sec @ 508 bytes
    ConnOverhead = .002;        $ conn overhead
    MaxCIPConnections = 128;    $ no more than 128 total connections
    MaxConsumersPerMcast = 64;  $ 64 consumers per multicast connection
```

### **7-3.6.12 Event Enumeration Section**

The Event Enumeration section [EventEnum] associates specific event or status codes within a device to an international string. The events identified may represent any event, error(including error response codes) or other condition within a device.

Events may be enumerated by the EventN entry keyword(s). There may be as many EventN entry keywords as necessary to identify all of the errors needing association.

**Table 7-3.42 Fields in the EventN entry keyword**

Field Name	Field Number	Data Type	Required/Optional
Object Reference	1	EPAUTH	Required
Reserved	2	empty	
Event Code 1	3	UDINT	Required
Additional Code 1	4	UDINT	Optional
Associated String 1	5	STRINGI	Required
Event Code 2	6	UDINT	Optional
Additional Code 2	7	UDINT	Optional
Associated String 2	8	STRINGI	Conditional
...			
Event Code J	(J-1)*3 +3	UDINT	Optional
Additional Code J	(J-1)*3 +4	UDINT	Optional
Associated String J	(J-1)*3 +5	STRINGI	Conditional

Where:

- Object Reference – This field shall only contain one of the following:
  - an EPAUTH identifying the object instance address (only Logical encoding of object class and explicit instance number)
  - an EPAUTH identifying all instances of a specific class (only Logical encoding of object class and no instance specified)
  - a null entry

This object reference is used to associate the following Error/Additional code(s) with the identified object(s).

A null entry specifies any instance of any object. If this field is null, then (1) the Additional Code field shall also be null and (2) the Event Code shall be within the non-object specific range.

When this field is not null it identifies some specific object instance (or object instances) within the device. The absence of an instance number specification indicates that the event enumerations pertain to any instances (including instance zero) of the related object.

- Reserved – This field shall be entered as a null field. This field is reserved for future expansion of the EventN entry keyword.
- Event Code n – This field specifies an Event Code that is combined with the Object Reference and Additional Code fields to form a unique event identification.

- Additional Code n – This field specifies an additional (code qualification) value that is combined with the Object Reference and the Event Code fields to form a unique event identification. The presence of a null field indicates any value is acceptable to form a value match.
- Associated String n – This international string is associated with the error identification. This field is required when Event Code n is specified.

The matching of an actual event condition (from an instance of a class and associated event and additional code) with a specific EventN entry keyword is dependant upon an application performing matching activities in the following order:

1. Actual error condition exactly matches an EventN entry keyword that explicitly specifies class and instance.
2. Actual error condition exactly matches an EventN entry keyword that explicitly specifies class and defaults to any instance (including instance zero).
3. Actual event condition's Event code matches an EventN entry keyword with null Object Reference (e.g. no class or instance specified) and exact match for Event code.

An example of an [EventEnum] section is shown below:

```

Event1 = "20 20 24 02",    $ Object identification: Analog Input Group
                           $ Instance 2 only
                           $ Reserved Field
                           $ ---
                           $ Error code 0x1F, Minor 0x00
0x1F, 0x00,                 $ Vendor specific error w/ additional code 0
{2,
 {"eng",0xD0,4,"Message text 1" },
 {"deu",0xD0,4,"Anzeigetext 1" } $
},

0x1F, 0x05,                 $ Vendor specific error w/ additional code 5
{ 2,
 {"eng",0xD0,4,"Message text 2" },
 {"deu",0xD0,4,"Anzeigetext 2" } $
},

0x0C, 0xFF,                 $ Object state conflict translation
{ 2,
 {"eng",0xD0,4,"Message text 3" },
 {"deu",0xD0,4,"Anzeigetext 3" } $
},

0x10, 0x00,                 $ Device state conflict translation
{ 2,
 {"eng",0xD0,4,"Message text 4" },
 {"deu",0xD0,4,"Anzeigetext 4" } $
},

0x10, 0x01,                 $ Another device state conflict translation
{ 2,
 {"eng",0xD0,4,"Message text 5" },
 {"deu",0xD0,4,"Anzeigetext 5" } $
};

Event2 = "20 01 24 03",    $ Instance 3 of identity
                           $ Reserved Field
0x10, 0x04,
{ 2,
 {"eng",0xD0,4,"Subnet communication in progress - Try again later" },
 {"deu",0xD0,4,"Subnetzkommunikation läuft - Versuchen Sie es noch einmal
später" } $
};

Event4 = "20 02",           $ Message Router (any instance)
                           $ Reserved Field
0x10, 0x01,
{ 2,
 {"eng",0xD0,4,"EEPROM update in progress - Try again later" },
 {"deu",0xD0,4,"EEPROM Update läuft - Versuchen Sie es noch einmal
später" } $
};

0x10, 0x02,
"{
 {"eng",0xD0,4,"Failed to initialize analog channel(s)" },

```

```

        {"deu", 0xD0, 4, "Analogkanal(-kanäle) konnte(n) nicht initialisiert
werden"} $ 
    },
    0x10, 0x04,
    {
    2,
    {"eng", 0xD0, 4, "Remote communication channel active"}, 
    {"deu", 0xD0, 4, "Remotekommunikationskanal aktiv"} $ 
};

Event5 = "",                      $ Any Object, Any instance
,                                $ Reserved Field
0x1F, ,                          $ (Note - Null add code also)
{
2,
{"eng", 0xD0, 4, "Subnet devices unresponsive"}, 
 {"deu", 0xD0, 4, "Geräte im Subnetz antworten nicht"} $ 
};

```

### 7-3.6.13 Symbolic Translation Section

The [SymbolicTranslation] section is used to publicize the translation between a Symbolic Segment or an ANSI Extended Symbol Segment encoded EPATH specification to the equivalent ParamN or AssemN entry keywords. Use of this section is required when a device utilizes a symbolic segment form of encoding within the EDS file.

The entry keyword for all symbolic translations shall consist of the character array, "Symbolic", combined with a decimal number, forming a unique keyword.

**Table 7-3.43 Fields in the Symbolic Keyword**

Field Name	Field number	Data type	Required/Optional
Symbolic Definition	1	EPATH	Required – Shall be a single symbolic segment specification
Translation	2	ParamN or AssemN	Required – Shall be ParamN or AssemN entry keyword reference

DeviceNet Specific Example:

```

[IO_Info]
Default = 0x0001;                      $ Poll connection is default
PollInfo = 0x0003, 1, 1;
StrobeInfo = 0x0003, 2, 0;

Input1 = 2, 16, 0x0001, "Specific volts", 2, "61 41";
Input2 = 8, 64, 0x0002, "HMI Data", 2, "61 42";
Output1 = 1, 8, 0x0001, "Motor Control", 7, "66 4D 6F 74 6F 72 34";

[SymbolicTranslation]
Symbolic1 = "61 41",                   $ The symbol "A" translates to
                                         $ item described by Param22
Symbolic2 = "61 42",                   $ The symbol "B" translates to
                                         $ collection described by Assem13
Symbolic3 = "66 4D 6F 74 6F 72 34",   $ The symbol "Motor4" is described by
                                         $ Assem56;                         $ Assem56 entry keyword

```

```
[Params]
...
Param22 = ,
6, "20 C6 24 03 30 14", $ Logical address within the device
0, 0xC7, 2, $ UINT
"Something vendor specific",
"Volts per meter", $ Units
"RF energy at equator";
...

Param8 = ,
6, "20 19 24 01 30 03", $ Run1 attribute in Control
0, 0xC1, 1, $ Supervisor
$ BOOL
"Run1", $ Name
"", $ Unitless
"When true, Motor runs forward";

Param9 = ,
6, "20 19 24 01 30 04", $ Run2 attribute in Control
0, 0xC1, 1, $ Supervisor
$ BOOL
"Run2", $ Name
"", $ Unitless
"When true, Motor runs in reverse";
...

[Assembly]
...
Assem13 = "HMI Specific Data",
, $ No equivalent logical path
8, $ Maximum size (8 bytes)
0, $ Descriptor
, $ Reserved fields
16,
16,
16,
16, ;
...

Assem56 = "Special Motor Control",
"20 04 24 CA 30 03", $ Accessible as Assembly 0xCA data attribute
1, $ 1 byte
0, $ Descriptor
, $ Reserved fields
1, Param8, $ First bit is Run1
1, Param9; $ Second bit is Run2
```

Network Unspecific Example:

```
...
[Assembly]
Assem1 = "Vendor Specific",
"20 04 24 01 30 03", $ In the device as assembly instance 1
1, $ 1 byte assembly size
0, $ Descriptor
, $ Reserved fields
```

```

4, "61 41",           $ 4 bits from symbolic path "A"
4, "61 42";          $ 4 bits from symbolic path "B"
...
[SymbolicTranslation]
Symbolic1 = "61 42",
"20 C7 24 02 30 11"; $ The symbol "B" translates to
                        $ Object 199, Instance 2, Attribute 17

Symbolic2 = "61 41",      $ The symbol "A" translates to
Param22;                 $ item described by Param22
...

```

### 7-3.6.14 Internationalization Section

The Internationalization section begins with the keyword [Internationalization] and shall describe EDS fields that can be internationalized. The table below shows the additional entries in the Internationalization section.

**Table 7-3.44 EDS Entry Keywords in the Internationalization Section**

Entry Name	Entry Keyword	Required/Optional
Descriptive Text	DescText	Optional
Device Type String	ProdTypStr	Optional
Product Name	ProdName	Optional
Parameter	ParamN, ProxyParamN, ProxiedParamN	Optional
Enumeration	EnumN	Optional
Group	GroupN	Optional
Assembly	AssemN, AssemExaN, ProxyAssemN, ProxyAssemExaN, ProxiedAssemN, ProxiedAssemExaN	Optional
Variant	VariantN, VariantExaN	Optional
Connection	ConnectionN	Optional
Port	PortN	Optional
Input	InputN, InputExaN	Conditional <sup>1</sup>
Output	OutputN, OutputExaN	Conditional <sup>1</sup>

1 – This keyword is not allowed if the device is not a DeviceNet device. If the device is a DeviceNet device, see Volume 3.

### **7-3.6.14.1 Descriptive Text Entry Keyword**

The “DescText” entry keyword is optional and shall contain the formatted field shown in Table 7-3.45.

**Table 7-3.45 DescText Keyword Format**

Field Name	Field Number	Data Type	Required/Optional
International Text	1	STRINGI	Required

#### **7-3.6.14.1.1 International Text, Field 1**

A single line of international text displayed by the configuration tool. The EDS developer assigns a meaningful line of text for this entry. This field value overrides the “DescText” keyword field value in the [File] section.

### **7-3.6.14.2 Device Type String Entry Keyword**

The “ProdTypeStr” entry keyword is optional and shall contain the formatted field shown in Table 7-3.46.

**Table 7-3.46 ProdTypStr Keyword Format**

Field Name	Field Number	Data Type	Required/Optional
International Type	1	STRINGI	Required

#### **7-3.6.14.2.1 International Type, Field 1**

International text description of device type as shown in Chapter 6-7. The string for vendor specific device types is at the vendors’ discretion. This field value overrides the “ProdTypStr” keyword field value in the [Device] section.

### **7-3.6.14.3 Product Name Entry Keyword**

The “ProdName” entry keyword is optional and shall contain the formatted field shown in Table 7-3.47.

**Table 7-3.47 ProdName Keyword Format**

Field Name	Field Number	Data Type	Required/Optional
International Name	1	STRINGI	Required

#### **7-3.6.14.3.1 International Name, Field 1**

International text product name. Same as Identity Object, Attribute 13 (if Identity Object, Attribute 13 is defined). When displayed, truncation may occur to meet the display capabilities. This field value overrides the “ProdName” keyword field value in the [Device] section.

#### **7-3.6.14.4 Parameter Entry Keywords**

The "ParamN", "ProxyParamN" and "ProxiedParamN" entry keywords are optional and shall contain the formatted fields shown in Table 7-3.48. These keywords correspond to the keywords of the same name in the [Params] section. When these keywords exist in the [Internationalization] section, fields 22, 23 and 24 for the corresponding ParamN, ProxyParamN or ProxiedParamN keyword in the [Params] section are overridden.

**Table 7-3.48 ParamN, ProxyParamN and ProxiedParamN Keywords Format**

Field Name	Field Number	Data Type	Required/Optional
International Parameter Name	1	STRINGI	Required
International Engineering Units	2	STRINGI	Required
International Help String	3	STRINGI	Required

##### **7-3.6.14.4.1 International Parameter Name, Field 1**

The international textual parameter name. When displayed, truncation may occur to meet the display capabilities.

##### **7-3.6.14.4.2 International Engineering Units, Field 2**

The international textual display units character array. If necessary, truncation of the retrieved text occurs to meet the maximum character array length allowed.

##### **7-3.6.14.4.3 International Help String, Field 3**

The international textual help character array. If necessary, truncation of the retrieved text occurs to meet the maximum character array length allowed.

#### **7-3.6.14.5 Enumeration Entry Keyword**

The "EnumN" entry keyword is optional and shall contain the formatted fields shown in Table 7-3.49.

If the tool supports the [Internationalization] section this "EnumN" keyword shall override the "EnumN" keyword in the [Params] section.

**Table 7-3.49 EnumN Keyword Format**

Field Name	Field Number	Data Type	Required/Optional
Value	1,3,5,...	data type of associated parameter	Field 1 Required, other fields optional
International Enumeration	2,4,6,...	STRINGI	Required if field N-1 is specified, not allowed otherwise.

##### **7-3.6.14.5.1 Value, Field 1, 3, 5,...**

The value to enumerate.

#### **7-3.6.14.5.2 International Enumeration, Field 2, 4, 6, ...**

The international string that enumerates the value in the preceding field.

#### **7-3.6.14.6 Group Entry Keyword**

The "GroupN" entry keyword is optional and shall contain the formatted field shown in Table 7-3.50.

**Table 7-3.50 DescText Keyword Format**

Field name	Field Number	Data type	Required/Optional
Internationalized Group Name String	1	STRINGI	Required

#### **7-3.6.14.6.1 Internationalized Group Name String, Field 1**

The international textual group name. This field value overrides the corresponding "GroupN" keyword name field value in the [Group] section.

#### **7-3.6.14.7 Assembly Entry Keywords**

The "AssemN", "AssemExaN", "ProxyAssemN", "ProxyAssemExaN", "ProxiedAssemN" and "ProxiedAssemExaN" entry keywords are optional and shall contain the formatted fields shown in Table 7-3.51. These keywords correspond to the keywords of the same name in the [Assembly] section.

**Table 7-3.51 AssemN, AssemExaN, ProxyAssemN, ProxyAssemExaN, ProxiedAssemN and ProxiedAssemExaN Keywords Format**

Field name	Field Number	Data type	Required/Optional
Internationalized Assembly Name	1	STRINGI	Required

#### **7-3.6.14.7.1 International Assembly Name, Field 1**

The international textual assembly name. This field value overrides the corresponding "AssemN", "AssemExaN", "ProxyAssemN", "ProxyAssemExaN", "ProxiedAssemN" and "ProxiedAssemExaN" keyword name field value in the [Assembly] section.

#### **7-3.6.14.8 Variant Entry Keywords**

The "VariantN" and "VariantExaN" entry keywords are optional and shall contain the formatted fields shown in Table 7-3.52. These keywords correspond to the keywords of the same name in the [Assembly] section.

**Table 7-3.52 VariantN and VariantExaN Keywords Format**

Field name	Field Number	Data type	Required/Optional
Internationalized Variant Name	1	STRINGI	Required
Internationalized Variant Help String	2	STRINGI	Required

#### **7-3.6.14.8.1 International Variant Name, Field 1**

The international textual variant name. This field value overrides the corresponding "VariantN" and "VariantExaN" keyword name field value in the [Assembly] section.

#### **7-3.6.14.8.2 International Variant Help String, Field 2**

The international textual variant help string. This field value overrides the corresponding "VariantN" and "VariantExaN" keyword help string field value in the [Assembly] section.

#### **7-3.6.14.9 Connection Entry Keyword**

The "ConnectionN" entry keyword is optional and shall contain the formatted fields shown in Table 7-3.53.

The "ConnectionN" keyword corresponds to the keywords of the same name in the [Connection Manager] section.

**Table 7-3.53 ConnectionN Keyword Format**

Field name	Field Number	Data type	Required/Optional
Internationalized Connection Name	1	STRINGI	Required
Internationalized Connection Help String	2	STRINGI	Required

#### **7-3.6.14.9.1 International Connection Name, Field 1**

The international textual connection name. This field value overrides the corresponding "ConnectionN" keyword name field value in the [Connection Manager] section.

#### **7-3.6.14.9.2 International Connection Help String, Field 2**

The international textual connection help string. This field value overrides the corresponding "ConnectionN" keyword help string field value in the [Connection Manager] section.

#### **7-3.6.14.10 Port Entry Keyword**

The "PortN" entry keyword is optional and shall contain the formatted field shown in Table 7-3.54.

**Table 7-3.54 PortN Keyword Format**

Field name	Field Number	Data type	Required/Optional
Internationalized Port Name	1	STRINGI	Required

#### **7-3.6.14.10.1 International Port Name, Field 1**

The international textual port name. This field value overrides the corresponding "PortN" keyword name field value in the [Port] section.

## **7-3.7 Modular EDS File Requirements**

This section describes the concept and contents of a Modular EDS and specifies the usage requirements.

### **7-3.7.1 Modular Section**

The [Modular] section begins with the keyword [Modular] and shall describe a chassis based system. The two types of modular devices shall be:

- Chassis;
- Module.

A [Modular] section that describes a chassis shall contain a required keyword “**DefineSlotsInRack**” as shown in Figure 7-3.22. The single field on this entry shall be a 16-bit unsigned integer (UINT) indicating the number of slots in the chassis. Even though an electronic key is defined for the chassis, it need not be addressable from the link. The SLOT keyword used in path definitions in the [Connection Manager] section shall range from 0 to the number of slots minus 1. The keyword “**SlotDisplayRule**” is optional. The single field on this entry shall be a parameter from the [Params] section (ParamN), which defines the translation between internal and external slot number.

**Figure 7-3.22 [Modular] section describing a chassis**

```
[File]
DescText = "Wonder Chassis EDS file";
CreateDate = 09-01-1997;
CreateTime = 17:23:00;
Revision = 1.1;
[Device]
VendCode = 65535;
VendName = "Widget Works, Inc.";
ProdType = 101;
ProdTypeStr = "Widget Works Generic";
ProdCode = 1;
MajRev = 1;
MinRev = 1;
ProdName = "Widget Chassis";
Catalog = "1234-chassis";
[Params]
Param1 =
0,                                $ first field shall equal 0
,,                                $ path size, path
0x0006,                            $ descriptor
0xC6,                               $ data type: Unsigned 8-bit Integer
1,                                $ data size in bytes
"Slot Naming Convention",          $ name
",",                               $ units
",",                               $ help string
0,4,0,                            $ min, max, default data values
0,0,0,0,                            $ mult, dev, base, offset scaling
0,0,0,0,                            $ mult, dev, base, offset link not used
0;                                 $ decimal places not used
Enum1 =
0,"n/a",1,"0",2,"1",3,"2",4,"3";
```

```
[Modular]
  DefineSlotsInRack = 5;
  SlotDisplayRule = Param1;
```

A [Modular] section that describes a module shall contain the "Width" and "Rack" entries.

The required entry with the keyword "**Width**" shall have two fields. The first field is required and indicates how many slots of the chassis are consumed by the module. The field shall be a 16-bit unsigned integer (UINT). The second field is optional and indicates which slot of the module connects to the chassis. If this optional field is omitted, slot 0 of the module is the connecting slot. The field shall be a 16-bit unsigned integer (UINT).

The entry keyword for all chassis, into which the module can be placed, shall consist of the character array, "Rack", combined with a decimal number. The numbers shall start at 1 for the first chassis, and shall be incremented for each additional chassis. Commas shall separate all fields, and a semicolon shall indicate the end of the entry. The fields for the "Rack" entries shall be as shown in Table 7-3.55.

**Table 7-3.55 Field for the Rack entry keyword**

Field name	Field number	Data type	Required/Optional
Vendor ID	1	UINT	Required
Product Type	2	UINT	Required
Product Code	3	UINT	Required
Major Revision	4	USINT	Required
Minor Revision	5	USINT	Required
reserved	6, 7, 8	empty	not used
Legal Slot	9, 10, 11 ...	UINT	Required

The "Vendor ID", "Product Type", "Product Code", "Major Revision" and "Minor Revision" field shall identify the electronic key of the chassis into which the module may be placed. The reserved field shall be empty. The "Legal Slot" fields shall identify the slots into which the module may be placed. The EDS for the module shall contain one "Rack" entry for each chassis into which the module may be placed as shown in Figure 7-3.23.

**Figure 7-3.23 [Modular] Section Describing a Module With a Network Connection**

## **7-3.7.2 Modular Additions to Basic EDS Sections**

### **7-3.7.2.1 CIP Path Addition for Modular EDS Files**

A CIP Path in a Modular EDS file may also contain the other following references (in addition to the references specified in the CIP Path definition):

- the keyword SLOT;
- the keyword SLOT\_MINUS\_ONE;

The keyword, SLOT, shall always evaluate to a USINT. The values substituted for the SLOT keyword shall correspond to the position of a module in a back plane. The keyword, SLOT\_MINUS\_ONE, shall always evaluate to a USINT. The values substituted for the SLOT\_MINUS\_ONE keyword shall correspond to the position of a module in the back plane minus 1.

Path Ex. 1: “20 04 24 SLOT 30 03”

If the module position is slot 4 the CIP resulting CIP path would be “20 04 24 04 30 03”.

Path Ex. 2: “20 04 24 SLOT\_MINUS\_ONE 30 03”

If the module position is slot 4 the CIP resulting CIP path would be “20 04 24 03 30 03”.

### **7-3.7.2.2 Additions to the Modular Section**

A [Modular] section that describes a module that does not have a link addressable Identity object may contain the entry keyword "ExternalID". The keyword shall have a single field. This field shall be a byte string that identifies the module. Modules that may be placed into slot 0 shall have an addressable Identity object and shall have no "ExternalID" entry.

A [Modular] section that describes a module that has a link connection and can be placed in slot 0 may contain the keyword "**GenericID**". The keyword shall have a single value. This field shall be a byte string that shall be included in the data segment for a module connection in place of the ExternalID when no module keying is desired.

A [Modular] section that describes a module that has a link connection and can be placed in slot 0 may contain the keyword "**ExternalIDExactMatch**". The keyword shall have a single value, Yes or No. Yes shall indicate that the ExternalID specifies one specific device, No shall indicate that the ExternalID specifies one of a set of compatible devices. If the "**ExternalIDExactMatch**" does not exist the default condition shall be that the ExternalID specifies one specific device.

A [Modular] section that describes a module that has a link connection and can be placed in slot 0 may contain the entry keyword “**Query**”. This keyword shall have 4 fields. The first field shall be a path that identifies a link addressable attribute that contains an array of external identifiers: one for each slot in the chassis except slot 0. The second field shall be the service to use with the query path (i.e. 1 – get attribute all or 14 – get attribute single). The third field shall be an integer that determines the number of bytes used to identify each module and shall be in the range 1 to 16. If a double slot module is in the chassis, the external identifier for the module shall appear twice in the array returned from a query. A query shall only be addressed to a module in slot 0. The fourth field shall be the ExternalID returned when an empty slot exists.

**Figure 7-3.24 [Modular] Section Describing a Module Without an Identity Object**

```
[Modular]
Width = 1;
Rack1 =
65535, 101, 1, 1, 1, ...,           $ this module can plug into
1, 2, 3, 4;                      $ slots 1, 2, 3 and 4 of
                                    $ this five slot chassis

Rack2 =
65535, 101, 2, 1, 1, ...,           ...
1, 2, 3, 4, 5, 6, 7;

Query = "20 04 24 07 30 03",1,2,"FF FF";
ExternalID = "12 34";
GenericID = "00 00";
ExternalIDExactMatch = No;
```

### 7-3.7.2.3 Additions to the Parameter Section

The “**ProxyParam**” and “**ProxiedParam**” keywords shall be used to describe parameters that are proxied by a CIP adapter device to another device that does not support the CIP protocol. An example of this is a ControlNet adapter module (the device proxying the connection) in a multiple slot I/O rack with an analog I/O module (the device the connection is proxied for).

The “**ProxyParam**” shall exist in the EDS for the device that performs the proxy.

The “**ProxiedParam**” keyword shall exist in the EDS for the device that the proxy is performed for.

The information in the [Modular] section shall be used to associate EDS files containing “**ProxyParam**” keywords to EDS files containing “**ProxiedParam**” keywords. This association shall exist when both EDS files specify a matching Rack entry.

The decimal number (that is combined with “**ProxyParam**” and “**ProxiedParam**”) shall be used to match a “**ProxyParam**” to a “**ProxiedParam**”. The field values of a matched “**ProxyParam**” and “**ProxiedParam**” pair shall be combined to constitute the same field value information that exists in a single “**Param**” entry. This combination shall be done by using the field value from the “**ProxyParam**” unless that field value is the keyword “**Module**”. When the field value specified in the “**ProxyParam**” is “**Module**” the field value specified in the “**ProxiedParam**” shall be used. It shall be legal to specify field values for “**ProxiedParam**” entries whose corresponding field value in the “**ProxyParam**” is not “**Module**”, however, these field value shall not be used, they shall exist only for documentation.

Another keyword may also exist in the [Params] section. This keyword shall be used to provide minimum, maximum and default values to be added to the “**ProxyParam**” minimum, maximum and default values. This entry keyword shall be “**ProxyParamSizeAdder**”, combined with the decimal number from the corresponding “**ProxyParam**” entry. Commas shall separate all fields, and a semicolon shall end the entry. Each “**ProxyParam**” entry shall consist of a Minimum Value, Maximum Value and Default Value fields. The definition of these fields matches the “**Param**” definitions. The “**ProxyParamSizeAdder**” keyword provides a means for an adapter on a module connection (“**ProxyConnect**”) to add adapter data to the module data and return the combined data on the connection.

Another keyword may also exist in the [Param] section that corresponds to the “**ProxyParam**”, “**ProxyEnum**”. “**ProxyEnum**” has the same definition as “**Enum**” except it is associated with “**ProxyParam**” instead of “**Param**”. A second keyword may also exist in the [Param] section that corresponds to the “**ProxiedParam**”, “**ProxiedEnum**”. “**ProxiedEnum**” has the same definition as “**Enum**” except it is associated with “**ProxiedParam**” instead of “**Param**”.

For bit fields specified in proxy keyword fields, the binary value shall allow a character of M (in addition to the 0 and 1 characters). The character M shall indicate to use the character specified in the proxied keyword field.

For example, the descriptor bits for a ProxyAssemN whose value is 0b00000000000000001M shall indicate that the assembly is a fixed sized expanded rack assembly and the ProxiedAssemN defines the allow value edit setting.

#### **7-3.7.2.4      Additions to the Assembly Section**

The following entries are additional values allowed for the "Member Reference" field within the Assembly section:

- ExternalID;
  - InputSlotMask0;
  - InputSlotMask1;
  - OutputSlotMask0;
  - OutputSlotMask1;
  - ConfigSlotMask0;
  - ConfigSlotMask1.

The “**ExternalID**” keyword shall indicate the usage of the device “**ExternalID**” value in the case when device keying is desired or the “**GenericID**” value in the case when module keying is not desired.

The “**InputSlotMask0**” keyword shall indicate the location of the input slot mask in the assembly. The preceding size field shall be required. An input slot mask is an array of bits which represents inclusion or exclusion of a modules target to originator data in this adapter rack connection. Bit 0 in this array represents slot 0, bit 1 represents slot 1, etc.

The “**InputSlotMask1**” keyword shall indicate the location of the input slot mask in the assembly. The preceding size field shall be required. An input slot mask is an array of bits which represents inclusion or exclusion of a modules target to originator data in this adapter rack connection. Bit 0 in this array represents slot 1, bit 1 represents slot 2, etc.

The “**OutputSlotMask0**” keyword shall indicate the location of the output slot mask in the assembly. The preceding size field shall be required. An output slot mask is an array of bits which represents inclusion or exclusion of a modules originator to target data in this adapter rack connection. Bit 0 in this array represents slot 0, bit 1 represents slot 1,etc.

The “**OutputSlotMask1**” keyword shall indicate the location of the output slot mask in the assembly. The preceding size field shall be required. An output slot mask is an array of bits which represents inclusion or exclusion of a modules originator to target data in this adapter rack connection. Bit 0 in this array represents slot 1, bit 1 represents slot 2 etc.

The “**ConfigSlotMask0**” keyword shall indicate the location of the configuration slot mask in the assembly. The preceding size field shall be required. A configuration slot mask is an array of bits which represents inclusion or exclusion of a modules configuration data in the fwd\_open of this adapter rack connection. Bit 0 in this array represents slot 0, bit 1 represents slot 1, etc.

The “**ConfigSlotMask1**” keyword shall indicate the location of the configuration slot mask in the assembly. The preceding size field shall be required. A configuration slot mask is an array of bits which represents inclusion or exclusion of a modules configuration data in the fwd\_open of this adapter rack connection. Bit 0 in this array represents slot 1, bit 1 represents slot 2, etc.

The “**ExternalID**” combined with a number (decimal) keyword intended purpose is to allow individual device keying for adapter rack connections. It shall indicate the usage of the device “**ExternalID**” value in the case when device keying is desired or the “**GenericID**” value in the case when module keying is not desired. The decimal number specifies the slot of this “**ExternalID**”.

The “**ProxyAssem**”, “**ProxyAssemExa**”, “**ProxiedAssem**” and “**ProxiedAssemExa**” keywords shall be used to describe assemblies that are proxied by a CIP adapter device to another device that does not support the CIP protocol. An example of this is a ControlNet adapter module (the device proxying the connection) in a multiple slot I/O rack with an analog I/O module (the device the connection is proxied for).

The “**ProxyAssem**”/”**ProxyAssemExa**” keywords shall exist in the EDS for the device that performs the proxy; the “**ProxiedAssem**”/”**ProxiedAssemExa**” keywords shall exist in the EDS for the device that the proxy is performed for. The information in the [Modular] section shall be used to associate EDS files containing “**ProxyAssem**”/”**ProxyAssemExa**” keywords to EDS files containing “**ProxiedAssem**”/”**ProxiedAssemExa**” keywords. This association shall exist when both EDS files specify a matching Rack entry.

The decimal number (that is combined with “**ProxyAssem**”/“**ProxyAssemExa**” and “**ProxiedAssem**”/“**ProxiedAssemExa**”) shall be used to match a “**ProxyAssem**”/“**ProxyAssemExa**” to a “**ProxiedAssem**”/“**ProxiedAssemExa**”. The field values of a matched “**ProxyAssem**”/“**ProxyAssemExa**” and “**ProxiedAssem**”/“**ProxiedAssemExa**” pair shall be combined to constitute the same field value information that exists in a single “**Assem**”/“**AssemExa**” entry. This combination shall be done by using the field value from the “**ProxyAssem**”/“**ProxyAssemExa**” unless that field value is one of the keywords “**Module**” or “**ModuleMemberList**”. When the field value specified in the “**ProxyAssem**”/“**ProxyAssemExa**” is “**Module**” the field value specified in the “**ProxiedAssem**”/“**ProxiedAssemExa**” shall be used. The field value “**Module**” shall not be used for “Member Size” or “Member Reference” fields. “**ModuleMemberList**” shall only be used in place of a “Member Size” and “Member Reference” field pair. When the field value specified in the “**ProxyAssem**”/“**ProxyAssemExa**” is “**ModuleMemberList**” all “Member Size” and “Member Reference” fields specified in the “**ProxiedAssem**”/“**ProxiedAssemExa**” shall be used. It shall be legal to specify field values for “**ProxiedAssem**”/“**ProxiedAssemExa**” entries whose corresponding field value in the “**ProxyAssem**”/“**ProxyAssemExa**” is not “**Module**”, however, these field values shall not be used, they shall exist only for documentation.

**Figure 7-3.25 Example of ProxyParam and ProxyAssem**

```
[Params]
Param1 = 0,,,0x0010,0xC7,2," Target Error Codes",
        "", "", 0,0xFFFF,0,0,0,0,0,0,0,0,0,0;
ProxyParam1 = 0,,,0x0000,0xC7,2,"input size",
        "", "", Module,Module,Module,0,0,0,0,,0;
ProxyParamSizeAdder1 = 4,4,4;

[Assembly]
Assem1 = "connection input format",,,,
         32,Param1,
         ,ProxyAssem1,
         ,ProxyAssem2;
ProxyAssem1 = "real time input format", "20 7D 24 SLOT 30 0A",,,,
             ModuleMemberList;
ProxyAssem2 = "real time status format", "20 7D 24 SLOT 30 0B",,,,
             ModuleMemberList;

[Connection Manager]
ProxyConnect1 = 0x010100002, 0x44244401,
                2, 0, , 2, ProxyParam1, Assem1, , , , "Listen Only", "",
                "01 SLOT_MINUS_ONE 20 04 24 03 2C 04 2C 02";
```

**Figure 7-3.26 Example of ProxiedParam and ProxiedAssem**

```
[Params]
ProxiedParam1 = ,,,,"input size","","",0,0xC7,2,,,,;

[Assembly]
ProxiedAssem1 = "real time input format",,,;
ProxiedAssem2 = "real time status format",,,16,;

[Connection Manager]
ProxiedConnect1 = ,,,0,,,,;
```

For bit fields specified in proxy keyword fields, the binary value shall allow a character of M (in addition to the 0 and 1 characters). The character M shall indicate to use the character specified in the proxied keyword field.

For example, the descriptor bits for a ProxyAssemN whose value is 0b00000000000000001M shall indicate that the assembly is a fixed sized expanded rack assembly and the ProxiedAssemN defines the allow value edit setting.

### **7-3.7.2.5 Additions to the Connection Manager Section**

The “**ProxyConnect**” and “**ProxiedConnect**” keywords shall be used to describe connections that are proxied by a CIP adapter device to another device that does not support the CIP protocol. An example of this is a ControlNet adapter module (the device proxying the connection) in a multiple slot I/O rack with an analog I/O module (the device the connection is proxied for).

The “**ProxyConnect**” keyword entry shall exist in the EDS for the device that performs the proxy. In the example above, this would be the ControlNet adapter module.

The “**ProxyedConnect**” keyword entry shall exist in the EDS for the device that the proxy is performed for. In the example above, this would be the analog I/O module.

The information in the [Modular] section shall be used to associate EDS files containing “**ProxyConnect**” keywords to EDS files containing “**ProxiedConnect**” keywords. This association shall exist when both EDS files specify a matching **Rack** entry.

The decimal number (that is combined with “ProxyConnect” and “ProxiedConnect”) shall be used to match a “ProxyConnect” to a “ProxiedConnect”. The field values of a matched “ProxyConnect” and “ProxiedConnect” pair shall be combined to constitute the same field value information that exists in a single “Connection” entry. This combination shall be done by using the field values from the “ProxyConnect” except for those fields where the value is the keyword “Module”. In those cases, the field value specified in the associated “ProxiedConnect” shall be used. It shall be legal to specify field values for “ProxiedConnect” entries whose corresponding field value in the “ProxyConnect” entry is not “Module”, however, these field values shall not be used, they shall exist only for documentation. The field value for the “ProxyConnect” “connection name string” field shall not be “Module”, the “ProxyConnect” shall always specify the “connection name string”.

This page is intentionally left blank

# **Volume 1: Common Industrial Protocol (CIP<sup>TM</sup>) Specification**

## **Chapter 8: Physical Layer**

## Contents

8-1	Introduction.....	3
-----	-------------------	---

## **8-1      Introduction**

The CIP application layer can be used on a variety of network technologies. Each CIP network specification consists of two volumes. This volume is not currently specifying any common physical layer behavior. See the appropriate CIP network adaptation volume for physical layer behavior defined on that network.

This page is intentionally left blank

## **Volume 1: Common Industrial Protocol (CIP<sup>TM</sup>) Specification**

### **Chapter 9: Indicators and Middle Layers**

## **Contents**

9-1	Introduction.....	3
-----	-------------------	---

## **9-1      Introduction**

The CIP application layer can be used on a variety of network technologies. Each CIP network specification consists of two volumes. This volume is not currently specifying any common indicator or middle layer behavior. See the appropriate CIP network adaptation specification for indicator or middle layer behavior defined on that network.

This page is intentionally left blank

## **Volume 1: Common Industrial Protocol (CIP<sup>TM</sup>) Specification**

### **Chapter 10: Bridging and Routing**

## **Contents**

10-1	Introduction.....	3
10-2	CIP Routing .....	3
10-3	Object Addressing Space .....	3
10-4	CIP Routing Behavior.....	5
10-5	DeviceNet Modifications.....	7
10-5.1	Unconnected Send Service Request Modification .....	7
10-5.2	Unconnected Send Service Response Modification .....	8
10-6	Examples.....	9
10-6.1	Using the Unconnected Send Service for Explicit Messaging – Single Hop.....	9
10-6.2	Using the Unconnected Send Service For Explicit Messaging – Multiple Hop .....	12
10-6.3	Using the Forward Open Service to Open an I/O Messaging Connection.....	15
10-6.4	Using the Forward Close Service to Close an I/O Messaging Connection.....	18

## **10-1      Introduction**

The communication objects within Chapter 3, in particular services of the Connection Manager, provide bridging and routing between CIP subnets. This chapter will describe how bridging and routing is accomplished using those objects.

## **10-2      CIP Routing**

The Forward\_Open, Forward\_Close, and Unconnected\_Send services of the Connection Manager object allow for routing of messages and for the establishment of connections across bridges. Use of the Port Segment within the Connection Path parameter of the above services provides the routing information. The Port Segment encoding is described in Appendix C. Routing is ‘fixed path’ (data will always follow the same path); this is critical for insuring consistent I/O transaction times.

Each Port Segment represents a ‘hop’ in the path, from one subnet to another, and the CIP router removes its Port Segment from the path before forwarding the service to the next destination. The Port Segment tells the router which port (subnet) to send the message on and the node address of the destination node on that subnet. The final target receives a Connection Path with no Port Segments remaining.

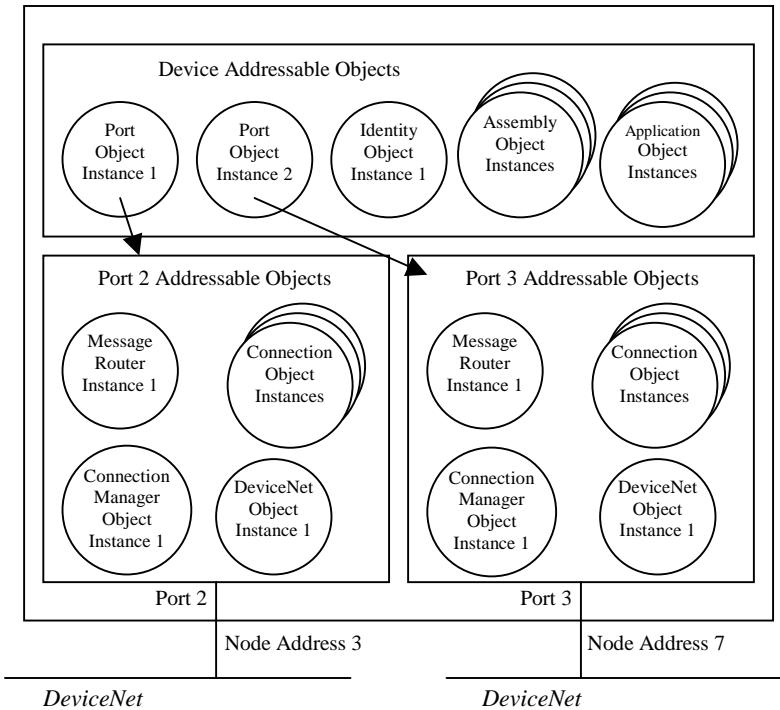
A connection originator can use the Port Object within each CIP router to determine the types of subnets, which can be reached (routed to) through that device. The Port Object, described in Chapter 3, provides the port name, port number within the device, and other port information for each routable port on the device.

## **10-3      Object Addressing Space**

A CIP router may actually be the Target Device in an Unconnected Send Request. One use for this would be to access link specific objects within the address space of port, which is not the port the request entered on. Link type objects (such as the ControlNet Object, DeviceNet Object, Connection Manager Object or Connection Object) within a device may (and in some cases are required to) have the same instance numbers on each port.

For example, considering a device containing multiple DeviceNet ports, requesting an attribute within instance one of the DeviceNet Object on a port will always return the value associated with the DeviceNet object on that port. Since the DeviceNet object associated with a different port is also addressed as instance one, it is necessary to ‘route’ to that object. The following diagram shows a possible internal object layout for a router. Note that in this example Port Object instance 1 is Port number 2 and Port Object instance 2 is Port number 3.

**Figure 10-3.1 Object Address Space within a CIP Node**



In this example, a request to Instance 1 of the DeviceNet Object entering on Port 2 (Node Address 3) will be serviced by the DeviceNet Object in Port 2's object space. In order to access Instance 1 of the DeviceNet Object in Port 3's object space, the Unconnected Send service of the Connection Manager Object for Port 2 needs to be used. The *Route\_Path* parameter within the Unconnected Send service would contain 03 07 indicating that the request should be routed to Port 3 within the device, Node 7 on that network. The router must detect that *it* is Node 7 on the network with which Port 3 is connected, process the request accordingly, and send a response.

## 10-4 CIP Routing Behavior

The table below presents the Events that a Connection Manager Object instance experiences and the appropriate actions to take.

**Table 10-4.1 CIP Routing Event/Action Matrix**

Event Description	Action to Take
Connection Manager Object Instance receives a valid Unconnected Send Request that DOES NOT need to be forwarded on through more intermediate hops.	Process and update the timing parameters <sup>1</sup> . Internally route the request to the specified port's protocol engine. Establish an Explicit Messaging Connection with the Remote Target Device. Extract the Explicit Messaging Request parameters from the Message Request portion of the Unconnected Send Service Data. Deliver the Explicit Messaging Request to the Remote Target Device. Start a “wait for response” timer whose value is the number of milliseconds specified by the current (updated) timing parameters.
Connection Manager Object Instance receives an invalidly formatted Unconnected Send Request.	Return a Routing Error that conveys the detected problem. See the Packet Validation section below for more details.
Failure to execute the Explicit Messaging transaction with the Remote Target Device. This is due to: <ul style="list-style-type: none"> <li>• Failure to establish an Explicit Messaging Connection</li> <li>• Open Explicit Messaging Connection Error Response</li> <li>• Timeout waiting for the Explicit Messaging Response.</li> </ul>	Return a Routing Error with the following settings: <ul style="list-style-type: none"> <li>• General Status field is set to 01</li> <li>• Additional Status array contains a single UINT value of 0x0204</li> <li>• remainingPathSize is set to the value zero (0).</li> </ul>
Failure to execute the Unconnected Send transaction with the next CIP Router in the path. This is due to: <ul style="list-style-type: none"> <li>• Failure to establish an Explicit Messaging Connection</li> <li>• Open Explicit Messaging Connection Error Response</li> <li>• Timeout waiting for the Unconnected Send Explicit Messaging Response.</li> </ul>	Return a Routing Error with the following settings: <ul style="list-style-type: none"> <li>• General Status field is set to 01</li> <li>• Additional Status array contains a single UINT value of 0x0204</li> <li>• Remaining Path Size is set to the number of words remaining in the “pre-stripped” path.</li> </ul>
Unconnected Send Request is received but it cannot be processed due to an internal resource error (e.g. queue overflow, no buffers available, etc.).	Return a Routing Error with the following characteristics: <ul style="list-style-type: none"> <li>• General Status field is set to 02</li> <li>• Additional Status array is empty</li> <li>• Remaining Path Size is set to the “pre-stripped” number of words remaining in the Route_Path field.</li> </ul>
Connection Manager Object Instance receives a valid Unconnected Send Request that needs to be routed through more intermediate hops before it reaches its final destination network.	Process and update the timing parameters <sup>1</sup> . Internally route the request to the specified port's protocol engine. Strip off the appropriate portion of the routing information and update the Transaction_Id field if necessary. Establish an Explicit Messaging Connection with the next CIP Router and deliver the updated Unconnected Send Request accordingly. Start a “wait for response” timer whose value is the number of milliseconds specified by the current timing parameters (AFTER they have been updated).

Event Description	Action to Take
Explicit Messaging Response OR an Unconnected Send Response is received by a CIP Router.	Determine whether or not the Explicit Messaging Connection across which the associated Unconnected Send Request is still active (did not experience an Inactivity/Watchdog timeout). Determine whether or not a timeout error has already been returned for this transaction. If the Explicit Messaging Connection is still active and a routing error has yet to be returned for this transaction, then formulate the appropriate Unconnected Send Response and return it across that Explicit Messaging Connection. This may entail restoring the requester's Transaction_Id field on DeviceNet. If a Routing Error has already been returned OR the Explicit Messaging Connection is no longer active, then the CIP Router shall discontinue all processing associated with this transaction.
CIP Router experiences a timeout while waiting for an Explicit Messaging Response from either another CIP Router (to which an Unconnected Send request had been forwarded) or the Remote Target Device (to which the actual Explicit Messaging Request had been delivered).	<p>Return a Routing Error with the following settings:</p> <ul style="list-style-type: none"> <li>• General Status field is set to 01</li> <li>• Additional Status array contains a single UINT value of 0x0204</li> <li>• remainingPathSize is set to the number of words remaining in the “pre-stripped” path.</li> </ul>

<sup>1</sup> – Any time a Connection Manager Object receives an Unconnected Send Request it SHALL process the timing parameters.

<sup>2</sup> – Whenever an Unconnected Send Request/Response is transmitted/received, a counter within the Connection Manager Object Instance associated with the port across which this transmission/reception took place shall be incremented.

## **10-5 DeviceNet Modifications**

The services of the Connection Manager are not supported by DeviceNet nodes when it is the target of the request. Thus, only nodes which provide routing to/from DeviceNet support the Connection Manager object. When the target node of a message is on DeviceNet, these routers are required to translate the message into normal DeviceNet explicit messaging format. As a result, DeviceNet nodes require no changes to accept routed messages from other CIP subnets.

In order to allow these routers the ability to handle multiple outstanding transactions on DeviceNet, the Unconnected Send service of the Connection Manager is modified when traversing a DeviceNet subnet. The details are provided in 10-5.1 and 10-5.2 below.

### **10-5.1 Unconnected Send Service Request Modification**

When sent on a DeviceNet subnet, the Unconnected Send service data is prepended with a 16-bit transaction ID. This transaction identifier is generated by the requesting device and returned by the router along with the response from the target. The modified service request is as shown below.

**Table 10-5.1 Unconnected Send Service Parameters**

Parameter Name	Data Type	Description	
Transaction_ID	UINT	Used for transaction management in the DeviceNet Requesting Device and DeviceNet Routers. This field is used for transaction matching by message originators on DeviceNet not passed on to other subnets	
Priority/Time_tick	BYTE	Used to calculate request timeout information.	
Time-out_ticks	USINT	Used to calculate request timeout information.	
Message_Request_Size	UINT	Specifies the number of bytes in the Message Request.	
Message_Request <sup>1</sup>	Struct of		
	Service	USINT	Service code of the request.
	Request_Path_Size	USINT	The number of 16 bit words in the Request_Path field (next element).
	Request_Path	Padded EPATH	This is an array of bytes whose contents convey the path of the request (Class ID, Instance ID, etc.) for this transaction.
	Request_Data	Array of octet	Service specific data to be delivered in the Explicit Messaging Request. If no additional data is to be sent with the Explicit Messaging Request, then this array will be empty.
Pad	USINT	Only present if Message_Request_Size is an odd value.	
Route_Path_Size	USINT	The number of 16 bit words in the Route_Path field.	
Reserved	USINT	Reserved byte. Shall be set to zero (0).	
Route_Path	Padded EPATH	Indicates the route to the Remote Target Device.	

<sup>1</sup> This is the Message Router Request Format as defined in Chapter 2.

## 10-5.2 Unconnected Send Service Response Modification

A DeviceNet router shall always return a successful response to an Unconnected Send service request. This success response may actually indicate that an error was encountered. This is necessary because additional error information is needed to handle routing errors and this additional information does not conform to the Error Response format defined for DeviceNet. As specified in Chapter 3, the Service code returned may be either the service code sent inside the Unconnected Send service data or the Unconnected Send service code. The 0x94 Error Response Service is never returned.

The modified successful and unsuccessful service responses are as shown below. Note that this is the data placed within the DeviceNet Service Data area; the reply service code is earlier in the packet.

**Table 10-5.2 Successful Unconnected Send Response**

Parameter Name	Data Type	Description
Transaction_ID	UINT	Echoes the value received in the associated Unconnected Send Request.
General Status	USINT	This value is zero (0) for successful transactions.
Reserved	USINT	Shall be zero.
Service Response Data	Array of byte	This field contains the Explicit Messaging Service Data returned by the Target Device/Object. For example, this would contain Attribute Data in response to a Get_Attribute_Single request. If the Explicit Messaging response returned by the Target Device/Object did not contain any Service Data, then this field shall be empty.

The response Service Data associated with an unsuccessful Unconnected Send response is defined below.

**Table 10-5.3 Unsuccessful Unconnected Send Response**

Parameter Name	Data Type	Description
Transaction_ID	UINT	Echoes the value received in the associated Unconnected Send Request.
General Status	USINT	One of the General Status codes listed in Appendix B Error Codes. If a routing error occurred, it shall be limited to the values specified in the Routing Error Values table.
Size of Additional Status	USINT	Number of 16 bit words in Additional Status array.
Additional Status	Array of UINT	When returning an error from a target, which is a DeviceNet node, the Additional Status shall contain the 8 bit Additional Error Code from the target in the lower 8 bits and a zero (0) in the upper 8 bits.
Remaining Path Size	USINT	This field is only present with routing type errors and indicates the number of words in the original route path ( <i>Route_Path</i> parameter of the Unconnected Send Request) as seen by the router that detects the error.

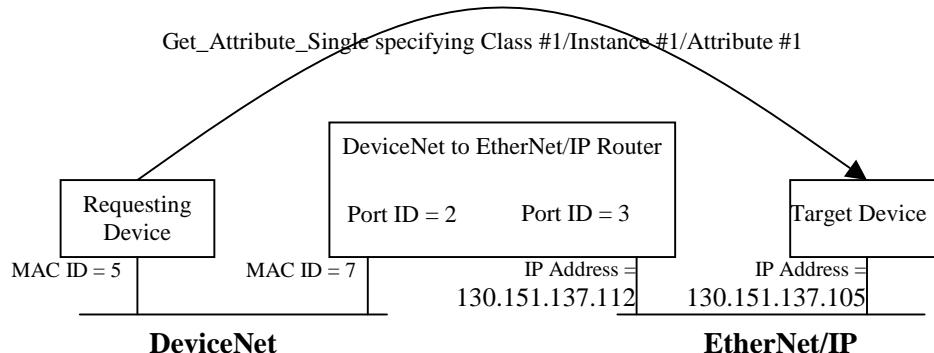
## 10-6 Examples

This section presents examples to further clarify the operation of the Connection Manager Object and the Unconnected Send service.

### 10-6.1 Using the Unconnected Send Service for Explicit Messaging – Single Hop

Assume that the *Requesting Device* wants to execute a Get\_Attribute\_Single on Class #1/Instance #2/Attribute ID #3 in the *Target Device*.

**Figure 10-6.1 Single Hop Unconnected Send (Explicit Message Request/Response)**



The table below presents the contents of the Service Data field for the Unconnected Send request which is used to execute the Get\_Attribute\_Single example noted above (all values are in hex).

**Table 10-6.2 Single Hop Unconnected Send Request Service Data**

Bytes	Meaning
01 00	The Requesting Device generated Transaction_Id field.
PP	Priority/Tick Time field.
tt	Connection Timeout Ticks field.
0C 00	Message_Size field = 12 bytes. Note that the Request_Data field is empty.
0E	Service field = Get_Attribute_Single.
05	Request_Path_Size field indicates that there are 5 words in the path field.
21 00 01 00 25 00 02 00 30 03	Path field specifying 16 bit Class Id = 1, 16 bit Instance ID = 2, 8 bit Attribute Id = 3. Note the presence of the pad bytes immediately following the 0x21 and 0x25 bytes. This path could have also been formatted as follows: 20 01 24 02 30 03 which makes use of 8 bit values to convey the Class and Instance ID data. In the 8-bit case, the pad bytes are not required.
09	Route_Path_Size field indicates that there are 9 words in the route path.
00	Reserved byte.
13 0F 31 33 30 2E 31 35 31 2E 31 33 37 2E 31 30 35 00	Contents of the Route_Path field. This field specifies the routing information. These bytes indicate that the request is to be delivered out Port #3 and to IP Address 130.151.137.105. This path encoding uses the Extended Link Address format for the IP address.

**Step 1 – Request delivered to the DeviceNet to EtherNet/IP Router**

The first step in the process calls for the Requesting Device to establish an Explicit Messaging Connection with the DeviceNet to EtherNet/IP Router and issue the Unconnected Send request. The table below presents the entire Unconnected Send request. Assume that the Message Body Format established for the Explicit Messaging Connection is DeviceNet 8/8.

**Table 10-6.3 Explicit Message Request on DeviceNet**

Bytes	Meaning
07	Frag = 0 <sup>1</sup> , Transaction ID = 0, Destination MACID = 7.
52	Service = 52. In this context, this specifies the Unconnected Send service.
06	Class ID of the Connection Manager Object Class.
01	The Instance ID Field specifying instance 1 of the Connection Manager.
01 00 pp tt 0C 00 0E 05 21 00 01 00 25 00 02 00 30 03 09 00 13 OF 31 33 30 2E 31 35 31 2E 31 33 37 2E 31 30 35 00	Unconnected Send Service Request Data. This is the previously described Unconnected Send service data describe above with the Transaction_Id field as the first UINT value (01 00 for this example).

<sup>1</sup> - This request would have to be fragmented to deliver it to the router. Fragmentation is not illustrated in this example.

The Unconnected Send request has now been delivered to the DeviceNet to EtherNet/IP Router via an Explicit Messaging Connection that exists between the Requesting Device and the DeviceNet to EtherNet/IP Router.

**Step 2 – Router delivers the request to the Target Device**

When the Connection Manager Object within the DeviceNet Router receives the Unconnected Send request it examines the contents of the *Route\_Path* field. In this case, the contents are “13 OF 31 33 30 2E 31 35 31 2E 31 33 37 2E 31 30 35 00”. This indicates that the next device in the hop is on Port #3 and its Node (IP) Address is 130.151.137.105. Additionally, since there is only a single Port/Node Address pair specified in the *Route\_Path*, the request has reached its destination network. When a DeviceNet Router receives an Unconnected Send Request that DOES NOT need to be forwarded through more intermediate CIP Routers, the following steps are taken:

- The DeviceNet Router executes the timing related logic associated with the *Priority/Tick Time* and *Connection Timeout Ticks* fields. If a timeout is detected, then a successful Unconnected Send response that specifies a Routing Error is returned to the Requesting Device.
- The DeviceNet Router extracts the actual transaction from the Message Request portion of the Service Data and delivers the Explicit Messaging Request to the Target Device. The actual method of delivering the explicit message is dependent on the link type.

In this example, the message request inside the Unconnected Send indicates a Get\_Attribute\_Single to Class #1, Instance #2, Attribute #3. This approach has no effect on the Target Device. The Target Device does not even realize it has just responded to a request that originated on a remote DeviceNet.

**Step 3 – Target returns the Explicit Message response to the Router**

The Target Device processes the explicit message and returns a response to the Router. When the Get\_Attribute\_Single Response is received by the DeviceNet Router it generates the Unconnected Send Response and internally delivers it to the Explicit Messaging Connection across which the Unconnected Send Request was originally received.

**Step 4 – Router returns the Unconnected Response to the Requesting Device**

The DeviceNet Router then delivers the Unconnected Send Response to the original Requesting Device.

Assume that the Target Device returns a successful response to the Get\_Attribute\_Single and the requested attribute is a UINT whose value is 0x1234. The text below presents the contents of the CAN Data Field associated with the Unconnected Send Response returned to the Requesting Device:

**Table 10-6.4 Successful Explicit Message Response on DeviceNet**

Bytes	Meaning
05	Frag = 0, Transaction ID = 0, Destination MAC = 5.
8E	Service = 8E. In this context, this specifies a response to the Unconnected Send service.
01 00	The Transaction_Id field echoed back in the response.
00	The General Status field. The value zero (0) indicates that there were no routing errors.
00	The Success status has no additional status.
34 12	The Target Device's Explicit Messaging Response Service Data field. In this case, the value 0x1234 was returned in the Get_Attribute_Single response.

Now assume that the Target Device returned an Error Response whose General Error Code was set to the value “2” and whose Additional Error Code field was set to the value “5”. The text below presents the contents of the CAN Data Field associated with the Unconnected Send Response returned to the Requesting Device. The absence of the *Remaining Path Size* field indicates that this error (Error Code = 2) was returned from the Target Device, not an intermediate router.

**Table 10-6.5 Unsuccessful Explicit Message Response on DeviceNet**

Bytes	Meaning
05	Frag = 0, Transaction ID = 0, Destination MAC = 5.
8E	Service = 0x8E. In this context, this specifies a response to the Unconnected Send service.
01 00	The Transaction_Id field echoed back in the response.
02	The General Status field. The non-zero value indicates there was an error.
01	The Size of Additional Status field. This indicates that the Target Device returned 16 bits of additional status.
05 00	This indicates that the Target Device returned an Additional Error Code of 0x05.

Finally, assume that the DeviceNet Router was unable to establish an Explicit Messaging Connection with the Target Device. In this case a Routing Error has occurred (the request was not delivered to the Target Device). The error code and extended status that most closely conveys the routing error is: General Code = 0x01, Extended Code = 0x0204 – *Unconnected Send timed out waiting for a response* (Note that error handling is dealt with in more detail in the Event/Action Matrix in 10-4). The text below presents the contents of the CAN Data Field associated with the Unconnected Send Response returned to the Requesting Device:

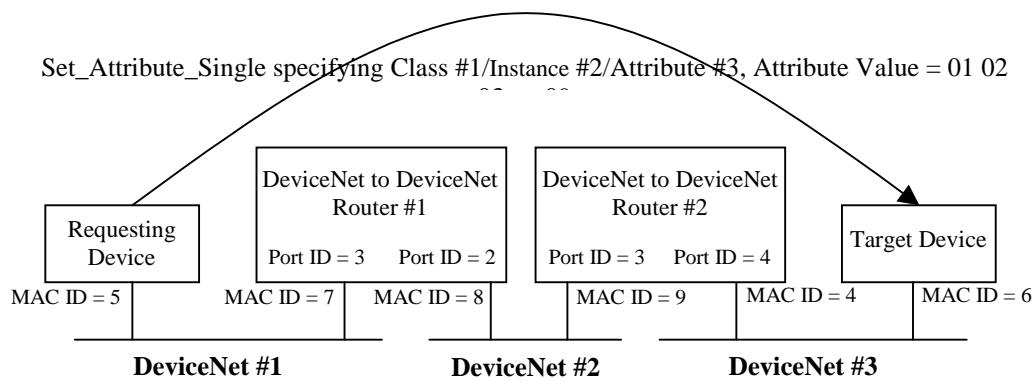
**Table 10-6.6 Routing Failure Explicit Message Response on DeviceNet**

Bytes	Meaning
05	Frag = 0, Transaction ID = 0, Destination MAC = 5.
D2	Service = D2. In this context, this specifies a response to the Unconnected Send service.
01 00	The Transaction_Id field echoed back in the response.
01	The General Status field. This indicates that a routing error was detected.
01	The Size of Additional Status field. This indicates that there is 16 bits of additional status.
04 02	The Additional Status field. There is a single UINT whose value is 0x0204.
00	The Remaining PathSize field. The value zero (0) indicates that an error was encountered during the attempt to establish communications with the Remote Target Device versus another DeviceNet Router.

## 10-6.2 Using the Unconnected Send Service For Explicit Messaging – Multiple Hop

Assume that the Requesting Device wants to execute a Set\_Attribute\_Single on Class #1/Instance #2/Attribute ID #3 in the Target Device. Assume that the data being sent to this attribute is a byte array whose value is 01 02 03 ... 09. Notice that this transaction will flow through two DeviceNet Routers.

**Figure 10-6.7 Multiple Hop Unconnected Send**



### **Step 1 – Request delivered to DeviceNet to DeviceNet Router #1**

The first step in the process calls for the Requesting Device to establish an Explicit Messaging Connection with DeviceNet to DeviceNet Router #1 and issue the Unconnected Send request. This is identical to Step #1 in the previous example. The text below presents the contents of the Unconnected Send request's Service Data field, which is used to execute the Set\_Attribute\_Single noted above (all values are in hex).

**Table 10-6.8 Multiple Hop Unconnected Send Request Service Data (First Hop)**

Bytes	Meaning
01 00	The Requesting Device generated Transaction_Id field.
07	Priority/Tick Time field (Time Tick = 128 ms).
0C	Connection Timeout Ticks field (12 Ticks, which results in 1536 ms timeout).
11 00	Message_Size field = 17 bytes. Note that this means that a pad byte will be inserted between the Request_Data field and the Route_Path_Size field.
10	Service field = Set_Attribute_Single.
03	Request_Path_Size field indicates that there are 3 words in the request path.
20 01 24 02 30 03	Request_Path field specifying 8 bit Class Id = 1, 8 bit Instance ID = 2, 8 bit Attribute Id = 3.
01 02 03 04 05 06 07 08 09	Request_Data field specifying the attribute data associated with the Set_Attribute_Single request.
00	Pad byte (required due to the odd length).
02	Route_Size field indicates that there are 2 words in the Route_Path.
00	Reserved byte.
02 09 04 06	Contents of the Route_Path field specifying the routing information. These bytes indicate that the request is to be delivered out Port #2 and to MAC ID 9 and then out Port #4 to the Target Device at MAC ID 6.

### **Step 2 – Request delivered to DeviceNet to DeviceNet Router #2**

When DeviceNet Router #1 receives the Unconnected Send request with this Service Data field it detects that there are more intermediate devices to go through before the Target Device can be reached. In this case, DeviceNet Router #1 executes the following steps:

- Executes the timing related logic associated with the *Priority/Tick Time* and *Connection Timeout Ticks* fields. If a timeout is detected, then a successful Unconnected Send response, which specifies a Routing Error, is returned to the Requesting Device.
- Strips its portion of the routing information from the Unconnected Send request's *Route\_Path* field and forwards the Unconnected Send Request accordingly. In this example, the "02 09" bytes constitute the routing information pertinent to DeviceNet Router #1 (next hop indicator). These bytes indicate that the Unconnected Send Request should be sent out Port #2 and to MAC ID #9 on that subnet.
- Establishes an Explicit Messaging Connection with DeviceNet Router #2 (if necessary) and sends the remaining data in the Unconnected Send service the second router.

The Service Data field of the Unconnected Send Request that DeviceNet Router #1 forwards to DeviceNet Router #2 is presented below. In this example, DeviceNet Router #1 is making use of the Transaction\_Id field for internal transaction management purposes and that the value “1” which was received with the request is already in use. In this case the DeviceNet Router #1 is free to allocate an unused value (assume the value “4”) and insert it into the packet it forwards.

**Table 10-6.9 Multiple Hop Unconnected Send Request Service Data (Second Hop)**

Bytes	Meaning
04 00	The Transaction_Id field inserted by the router.
07	Priority/Tick Time field (Time Tick = 128 ms).
08	Connection Timeout Ticks field (8 Ticks, which results in 1024 ms timeout, 512 ms less than was originally received).
11 00	Message_Size field = 17 bytes.
10	Service field = Set_Attribute_Single.
03	Request_Path_Size field indicates that there are 3 words in the request path.
20 01 24 02 30 03	Request_Path field specifying 8 bit Class Id = 1, 8 bit Instance ID = 2, 8 bit Attribute Id = 3.
01 02 03 04 05 06 07 08 09	Request_Data field specifying the attribute data associated with the Set_Attribute_Single request.
00	Pad byte.
01	Route_Size field indicates that there is 1 word in the Route_Path.
00	Reserved byte.
04 06	Route_Path indicating that the request is to be delivered out Port #4 to the Target Device at MAC ID 6. Note that DeviceNet Router #1 has <i>stripped</i> its routing information from the packet.

### **Step 3 – Request delivered to Remote Target Device**

When DeviceNet Router #2 receives this Service Data field in the Unconnected Send Request it behaves identical to the DeviceNet Router in the first example. Specifically, DeviceNet Router #2 executes the following steps:

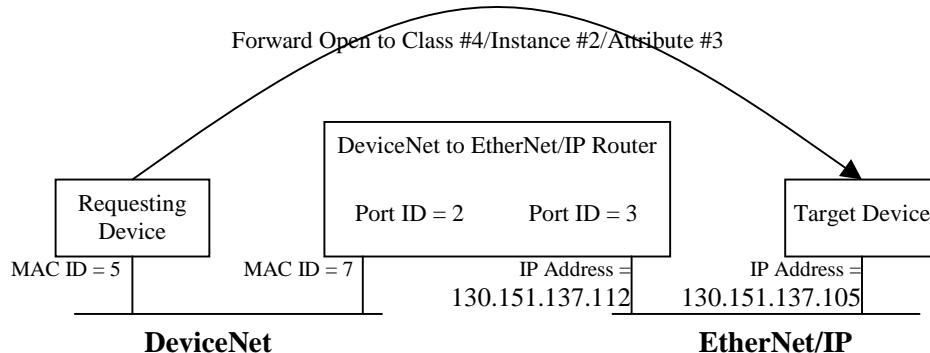
- Executes the timing related logic associated with the *Priority/Tick Time* and *Connection Timeout Ticks* fields. If a timeout is detected, then a successful Unconnected Send response that specifies a Routing Error is returned to the Requesting Device.
- Since there is no additional path routing, the device at MAC ID 6 is the Remote Target Device. The DeviceNet Router establishes an Explicit Messaging Connection with that device (if necessary), extracts the actual transaction from the Message Request portion of the Service Data, and delivers the Explicit Messaging Request to the Target Device.

The response returns back to the Requesting Device with each DeviceNet Router utilizing the Transaction\_Id field to facilitate internal transaction management. If an error was detected at any point in the process, then the appropriate error information would be returned in the successful Unconnected Send Response.

### 10-6.3 Using the Forward Open Service to Open an I/O Messaging Connection

Assume that the *Requesting Device* wants to make an I/O connection to an I/O Assembly. This data for this Assembly is identified as Class #4/Instance #2/Attribute ID #3 in the *Target Device*.

**Figure 10-6.10 Forward Open to Establish I/O Connection**



The table below presents the contents of the Service Data field for the Forward Open request which is used to execute the connection establishment example noted above (all values are in hex).

**Table 10-6.11 Forward Open Request Service Data**

Byte Value	Meaning
pp	Priority/Tick Time field.
tt	Connection Timeout Ticks field.
45 00 00 00	O_to_T Connection ID – Chosen by originating node (CAN ID 0x045 = Node 5, Group 1, Msg ID 1)
FF FF FF FF	T_to_O Connection ID – Chosen by target node
0C 00	Connection Serial Number
FF FF	Originator Vendor ID (Vendor ID = 65535)
00 01 02 03	Originator Serial Number (Serial Number = 0x03020100)
00	Connection Timeout Multiplier
00 00 00	Reserved
20 A1 07 00	O_to_T RPI (0x7A120 = 500 ms)
	O_to_T Connection Parameters
20 A1 07 00	T_to_O RPI (0x7A120 = 500 ms)
	T_to_O Connection Parameters
82	Transport Type / Trigger (Class 2 Cyclic Server)
0E	Connection_Path_Size field indicates that there are 14 words in the connection path.
13 0F 31 33 30 2E 31 35 31 2E 31 33 37 2E 31 30 35 21 00 04 00 25 00 02 00 30 03 00	Contents of the Connection_Path field. This field specifies the routing/connection information. These bytes indicate that the request is to be delivered out Port #3 and to IP Address 130.151.137.105. This path encoding uses the Extended Link Address format for the IP address. It further indicates that the application being connected to by specifying 16 bit Class ID = 4, 16 bit Instance ID = 2, 8 bit Attribute ID = 3.

**Step 1 – Request delivered to the DeviceNet to EtherNet/IP Router**

The first step in the process calls for the Requesting Device to establish an Explicit Messaging Connection with the DeviceNet to EtherNet/IP Router and issue the Forward Open request. The table below presents the entire Forward Open request. Assume that the Message Body Format established for the Explicit Messaging Connection is DeviceNet 8/8.

**Table 10-6.12 Forward Open Request on DeviceNet**

Bytes	Meaning
07	Frag = 0 <sup>1</sup> , Transaction ID = 0, Destination MACID = 7.
54	Service = 54. In this context, this specifies the Forward Open service.
06	Class ID of the Connection Manager Object Class.
01	The Instance ID Field specifying instance 1 of the Connection Manager.
01 00 pp tt 0C 00 0E 05 21 00 04 00 25 00 02 00 30 03 09 00 13 0F 31 33 30 2E 31 35 31 2E 31 33 37 2E 31 30 35 00	Forward Open Service Request Data. This is the previously described Forward Open service data described above.

<sup>1</sup> - This request would have to be fragmented on DeviceNet to deliver it to the router. DeviceNet fragmentation is not illustrated in this example.

The Forward Open request has now been delivered to the DeviceNet Router via an Explicit Messaging Connection that exists between the Originating Device and the DeviceNet Router.

**Step 2 – Router creates a CIP Bridged connection**

Upon receipt of a Forward Open to the Connection Manager object, the router shall (if it has the resources available) create a CIP Bridged connection (Class Code 0x05) and place it in the Configuring state. The remaining attributes are set with either the value received in the service request or the appropriate default values. If resources are unavailable, an error response is returned.

**Step 3 – Router creates connection with the Target Device**

After the Connection Manager Object within the DeviceNet Router processes the Forward Open request it examines the contents of the Connection\_Path field. In this case, the contents are “13 0F 31 33 30 2E 31 35 31 2E 31 33 37 2E 31 30 35 21 00 04 00 25 00 02 00 30 03 00”. This indicates that the next device in the hop is on Port #3 and its Node (IP) Address is 130.151.137.105. Additionally, since there is only a single Port/Node Address pair specified in the Connection\_Path field, the request has reached its destination network. When a DeviceNet Router receives a Forward Open Request that DOES NOT need to be forwarded through more intermediate CIP Routers, the following steps are taken:

- The DeviceNet Router executes the timing related logic associated with the Priority/Tick Time and Connection Timeout Ticks fields. If a timeout is detected, then a successful Forward Open response that specifies a Routing Error is returned to the Requesting Device.
- The DeviceNet Router creates an I/O connection with the Target Device. Once the I/O connection is created, the connection attributes are set, and the connection is applied. (Note that if the connection path indicates the Message Router object, then the router opens an Explicit Messaging connection using the UCMM.)

In the example above, the connection path inside the Forward Open indicates an I/O connection to Class #4, Instance #2, Attribute #3.

**Step 4 – Router returns the Forward Open Response to the Requesting Device**

The DeviceNet Router then delivers the Forward Open Response to the original Requesting Device. This response will contain the actual packet rates and the connection identifiers to be used.

The service data field for a Forward Open response is shown below.

**Table 10-6.13 Forward Open Response Service Data**

Byte Value	Meaning
45 00 00 00	O_to_T Connection ID – Chosen by originating node (CAN ID 0x045 = Node 5, Group 1, Msg ID 1)
07 01 00 00	T_to_O Connection ID – Chosen by target node (CAN ID 0x107 = Node 7, Group 1, Msg ID 4)
0C 00	Connection Serial Number
FF FF	Originator Vendor ID (Vendor ID = 65535)
00 01 02 03	Originator Serial Number (Serial Number = 0x03020100)
00	Connection Timeout Multiplier
00 00 00	Reserved
20 A1 07 00	O_to_T API (0x7A120 = 500 ms)
20 A1 07 00	T_to_O API (0x7A120 = 500 ms)
00	Application reply size (no application reply data)
00	Reserved

Assume that the connection to the Target Device was successful. The text below presents the contents of the CAN Data Field associated with the Forward Open Response returned to the Requesting Device:

**Table 10-6.14 Forward Open Response on DeviceNet**

Bytes	Meaning
07	Frag = 0, Transaction ID = 0, Destination MAC = 7.
D4	Service = D4. In this context, this specifies a response to the Forward Open service.
00	The General Status field. The value zero (0) indicates that there were no routing errors.
00	Reserved byte.
45 00 00 00 07 01 00 00 0C 00 FF FF 00 01 02 03 00 00 00 00 20 A1 07 00 20 A1 07 00 00 00	Forward Open Service Response Data. This is the previously described Forward Open service data described above.

## 10-6.4 Using the Forward Close Service to Close an I/O Messaging Connection

The table below presents the contents of the Service Data field for the Forward Close request to delete the connection, which was established in the example above (all values are in hex).

**Table 10-6.15 Forward Close Request Service Data**

Byte Value	Meaning
PP	Priority/Tick Time field.
tt	Connection Timeout Ticks field.
0C 00	Connection Serial Number
FF FF	Originator Vendor ID (Vendor ID = 65535)
00 01 02 03	Originator Serial Number (Serial Number = 0x03020100)
0E	Connection_Path_Size field indicates that there are 14 words in the connection path.
00	Reserved
13 0F 31 33 30 2E 31 35 31 2E 31 33 37 2E 31 30 35 21 00 04 00 25 00 02 00 30 03 00	Contents of the Connection_Path field. This field specifies the routing/connection information. These bytes indicate that the request is to be delivered out Port #3 and to IP Address 130.151.137.105. This path encoding uses the Extended Link Address format for the IP address. It further indicates the application being disconnected from by specifying 16 bit Class ID = 4, 16 bit Instance ID = 2, 8 bit Attribute ID = 3.

### Step 1 – Request delivered to the DeviceNet to EtherNet/IP Router

The first step in the process calls for the Requesting Device to establish an Explicit Messaging Connection with the DeviceNet to EtherNet/IP Router (if one does not exist) and issue the Forward Close request. The table below presents the entire Forward Open request. Assume that the Message Body Format established for the Explicit Messaging Connection is DeviceNet 8/8.

**Table 10-6.16 Forward Close Request on DeviceNet**

Bytes	Meaning
07	Frag = 0 <sup>1</sup> , Transaction ID = 0, Destination MACID = 7.
5E	Service = 5E. In this context, this specifies the Forward Close service.
06	Class ID of the Connection Manager Object Class.
01	The Instance ID Field specifying instance 1 of the Connection Manager.
01 00 pp tt 0C 00 0E 05 21 00 04 00 25 00 02 00 30 03 09 00 13 0F 31 33 30 2E 31 35 31 2E 31 33 37 2E 31 30 35 00	Forward Close Service Request Data. This is the previously described Forward Close service data described above.

<sup>1</sup>- This request would have to be fragmented to deliver it to the router. DeviceNet fragmentation is not illustrated in this example.

The Forward Close request has now been delivered to the DeviceNet Router via an Explicit Messaging Connection that exists between the Originating Device and the DeviceNet Router.

**Step 2 – Router deletes the CIP Bridged connection with Target**

After the Connection Manager Object within the DeviceNet Router processes the Forward Close request it examines the contents of the *Connection\_Path* field. In this case, the contents are “13 0F 31 33 30 2E 31 35 31 2E 31 33 37 2E 31 30 35 21 00 04 00 25 00 02 00 30 03 00”. This indicates that the next device in the hop is on Port #3 and its Node (IP) Address is 130.151.137.105. Additionally, since there is only a single Port/Node Address pair specified in the *Connection\_Path*, the request has reached its destination network. When a DeviceNet Router receives a Forward Close Request that DOES NOT need to be forwarded through more intermediate CIP Routers, the following steps are taken:

The DeviceNet Router executes the timing related logic associated with the *Priority/Tick Time* and *Connection Timeout Ticks* fields. If a timeout is detected, then a successful Forward Close response that specifies a Routing Error is returned to the Requesting Device.

The DeviceNet Router deletes the I/O or Explicit Messaging connection with the Target Device.

**Step 3 – Router returns the Forward Close Response to the Requesting Device and releases internal resources**

The DeviceNet Router then delivers the Forward Close Response to the original Requesting Device and releases all internal resources related to the connections on both the DeviceNet and EtherNet/IP subnets.

The service data field for a Forward Close response is shown below.

**Table 10-6.17 Forward Close Response Service Data**

Byte Value	Meaning
0C 00	Connection Serial Number
FF FF	Originator Vendor ID (Vendor ID = 65535)
00 01 02 03	Originator Serial Number (Serial Number = 0x03020100)
00	Application reply size (no application reply data)
00	Reserved

Assume that the connection to the Target Device was successful. The text below presents the contents of the CAN Data Field associated with the Forward Close Response returned to the Requesting Device:

**Table 10-6.18 Forward Close Response on DeviceNet**

Bytes	Meaning
07	Frag = 0, Transaction ID = 0, Destination MAC = 7.
DE	Service = DE. In this context, this specifies a response to the Forward Close service.
00	The General Status field. The value zero (0) indicates that there were no routing errors.
00	Reserved byte.
0C 00 FF FF 00 01 02 03 00 00	Forward Close Service Response Data. This is the previously described Forward Close service data described above.

This page is intentionally left blank

## **Volume 1: Common Industrial Protocol (CIP<sup>TM</sup>) Specification**

### **Appendix A: Explicit Messaging Services**

---

## Contents

A-1	Introduction.....	5
A-2	Service Definitions .....	5
A-3	CIP Common Services .....	6
A-4	CIP Common Service Definitions.....	7
A-4.1	Get_Attributes_All - Service Code: 01 <sub>Hex</sub> .....	7
A-4.1.1	Service Requirements .....	7
A-4.1.2	Request Service Data Field Parameters .....	7
A-4.1.3	Success Response Service Data Field Parameters .....	8
A-4.2	Set_Attributes_All - Service Code: 02 <sub>Hex</sub> .....	9
A-4.2.1	Service Requirements .....	9
A-4.2.2	Request Service Data Field Parameters .....	9
A-4.2.3	Success Response Service Data Field Parameters .....	9
A-4.3	Get_Attribute_List - Service Code: 03 <sub>Hex</sub> .....	10
A-4.3.1	Service Requirements .....	10
A-4.3.2	Request Service Data Field Parameters .....	10
A-4.3.3	Success Response Service Data Field Parameters .....	10
A-4.4	Set_Attribute_List - Service Code: 04 <sub>Hex</sub> .....	11
A-4.4.1	Service Requirements .....	11
A-4.4.2	Request Service Data Field Parameters .....	11
A-4.4.3	Success Response Service Data Field Parameters .....	11
A-4.5	Reset - Service Code: 05 <sub>hex</sub> .....	12
A-4.5.1	Service Requirements .....	12
A-4.5.2	Request Service Data Field Parameters .....	12
A-4.5.3	Success Response Service Data Field Parameters .....	12
A-4.6	Start - Service Code: 06 <sub>Hex</sub> .....	13
A-4.6.1	Service Requirements .....	13
A-4.6.2	Request Service Data Field Parameters .....	13
A-4.6.3	Success Response Service Data Field Parameters .....	13
A-4.7	Stop - Service Code: 07 <sub>Hex</sub> .....	14
A-4.7.1	Service Requirements .....	14
A-4.7.2	Request Service Data Field Parameters .....	14
A-4.7.3	Success Response Service Data Field Parameters .....	14
A-4.8	Create - Service Code: 08 <sub>Hex</sub> .....	15
A-4.8.1	Service Requirements .....	15
A-4.8.2	Request Service Data Field Parameters .....	15
A-4.8.3	Success Response Service Data Field Parameters .....	15
A-4.9	Delete - Service Code: 09 <sub>Hex</sub> .....	16
A-4.9.1	Service Requirements .....	16
A-4.9.2	Request Service Data Field Parameters .....	16
A-4.9.3	Success Response Service Data Field Parameters .....	16
A-4.10	Multiple_Service_Packet - Service Code: 0A <sub>Hex</sub> .....	17
A-4.10.1	Service Requirements .....	17
A-4.10.2	Request Service Data Field Parameters .....	17
A-4.10.3	Success Response Service Data Field Parameters .....	18
A-4.11	Apply_Attributes - Service Code: 0D <sub>Hex</sub> .....	19
A-4.11.1	Service Requirements .....	19
A-4.11.2	Request Service Data Field Parameters .....	19
A-4.11.3	Success Response Service Data Field Parameters .....	19
A-4.12	Get_Attribute_Single - Service Code: 0E <sub>Hex</sub> .....	20
A-4.12.1	Service Requirements .....	20
A-4.12.2	Request Service Data Field Parameters .....	20
A-4.12.3	Success Response Service Data Field Parameters .....	20

A-4.13	Set_Attribute_Single - Service Code: 10 <sub>Hex</sub> .....	21
A-4.13.1	Service Requirements .....	21
A-4.13.2	Request Service Data Field Parameters .....	21
A-4.13.3	Success Response Service Data Field Parameters .....	21
A-4.14	Find_Next_Object_Instance - Service Code: 11 <sub>Hex</sub> .....	22
A-4.14.1	Service Requirements .....	22
A-4.14.2	Request Service Data Field Parameters .....	23
A-4.14.3	Success Response Service Data Field Parameters .....	23
A-4.15	Restore - Service Code: 15 <sub>Hex</sub> .....	24
A-4.15.1	Service Requirements .....	24
A-4.15.2	Request Service Data Field Parameters .....	24
A-4.15.3	Success Response Service Data Field Parameters .....	24
A-4.16	Save - Service Code: 16 <sub>Hex</sub> .....	25
A-4.16.1	Service Requirements .....	25
A-4.16.2	Request Service Data Field Parameters .....	25
A-4.16.3	Success Response Service Data Field Parameters .....	25
A-4.17	No Operation (NOP) - Service Code: 17 <sub>Hex</sub> .....	26
A-4.17.1	Required Behavior .....	26
A-4.17.2	Request Service Data Field Parameters .....	26
A-4.17.3	Success Response Service Data Field Parameters .....	26
A-4.18	Get_Member - Service Code: 18 <sub>Hex</sub> .....	27
A-4.18.1	Service Requirements .....	27
A-4.18.2	Request Service Data Field Parameters .....	27
A-4.18.3	Success Response Service Data Field Parameters .....	27
A-4.19	Set_Member - Service Code: 19 <sub>Hex</sub> .....	28
A-4.19.1	Service Requirements .....	28
A-4.19.2	Request Service Data Field Parameters .....	28
A-4.19.3	Success Response Service Data Field Parameters .....	28
A-4.20	Insert_Member - Service Code: 1A <sub>Hex</sub> .....	29
A-4.20.1	Service Requirements .....	29
A-4.20.2	Request Service Data Field Parameters .....	29
A-4.20.3	Success Response Service Data Field Parameters .....	29
A-4.21	Remove_Member - Service Code: 1B <sub>Hex</sub> .....	30
A-4.21.1	Service Requirements .....	30
A-4.21.2	Request Service Data Field Parameters .....	30
A-4.21.3	Success Response Service Data Field Parameters .....	30
A-4.22	GroupSync - Service Code: 1C <sub>Hex</sub> .....	31
A-4.22.1	Service Requirements .....	31
A-4.22.2	Request Service Data Field Parameters .....	31
A-4.22.3	Response Service Data Field Parameters .....	31
A-5	Member Service Protocols .....	32
A-5.1	Member ID/EX Description.....	32
A-5.2	Member Request Message - Basic .....	33
A-5.3	Member Request Message - Extended .....	34
A-5.3.1	Multiple Sequential Members - Extended Protocol .....	35
A-5.3.2	International String Selection– Extended Protocol .....	36
A-6	CIP Encoding Examples .....	38
A-6.1	Example Object Class Definitions .....	38
A-6.2	Encoding Examples .....	39
A-6.2.1	Get_Attributes_All.....	40
A-6.2.2	Set_Attributes_All .....	41
A-6.2.3	Reset.....	42
A-6.2.4	Start.....	43
A-6.2.5	Stop .....	44
A-6.2.6	Create .....	45

A-6.2.7	Delete .....	46
A-6.2.8	Apply_Attributes.....	47
A-6.2.9	Get_Attribute_Single .....	48
A-6.2.10	Set_Attribute_Single.....	49
A-6.2.11	Find_Next_Object_Instance.....	50
A-6.2.12	Restore .....	51
A-6.2.13	Save.....	52
A-6.2.14	No Operation.....	53
A-6.2.15	Error Response.....	54
A-6.2.16	Get_Member .....	55
A-6.2.17	Set_Member .....	56
A-6.2.18	Insert_Member .....	57
A-6.2.19	Remove_Member.....	58

## **A-1      Introduction**

This appendix contains information about Explicit Messaging services. The CIP *Common Services* are defined and examples illustrating the encoding of CIP Common Services are provided. CIP Common Services are those services whose request/response parameters and required behaviors are defined in this document.

## **A-2      Service Definitions**

For any CIP service to be considered “fully defined,” its definition must include the following information:

- A brief functional definition explaining what the service does (why an object would request the service)
- Behaviors associated with the service
- Parameters which are placed in the Service Data Field of an Explicit Request Message, including:
  - Data type
  - Description
- Parameters which are placed in the Service Data Field of an Explicit Response Message, including:
  - Data type
  - Description

## A-3 CIP Common Services

The codes and names of the CIP Common Services are listed below.

**Table A-3.1 CIP Service Codes and Names**

Service Code (in hex)	Service Name
00	Reserved for future use
01	Get_Attributes_All
02	Set_Attributes_All Request
03	Get_Attribute_List
04	Set_Attribute_List
05	Reset
06	Start
07	Stop
08	Create
09	Delete
0A	Multiple_Service_Packet
0B-0C	Reserved for future use
0D	Apply_Attributes
0E	Get_Attribute_Single
0F	Reserved for future use
10	Set_Attribute_Single
11	Find_Next_Object_Instance
12 - 13	Reserved for future use
14	Error Response (used by DeviceNet only)
15	Restore
16	Save
17	No Operation (NOP)
18	Get_Member
19	Set_Member
1A	Insert_Member
1B	Remove_Member
1C	GroupSync
1D-31	Reserved for additional Common Services

## A-4 CIP Common Service Definitions

This section provides a general description of the CIP Common Services. Objects Classes/Instances that utilize these Services must provide a detailed description of their specific use.

### A-4.1 Get\_Attributes\_All - Service Code: 01<sub>Hex</sub>

Returns the contents of the instance or class attributes defined in the object definition.

#### A-4.1.1 Service Requirements

The following list details requirements associated with the Get\_Attributes\_All service:

1. The structure of the information in the successful response's Service Data Field adheres to the Get\_Attributes\_All response structure defined by the object or class. Support of this service requires the Object and/or Class to provide a detailed definition of the format of the data sent in the response message.
2. If the request is successfully serviced, then a Get\_Attributes\_All Response is returned. The Service Data Field of the response contains the attribute data. If an error is detected, then an Error Response is returned
3. A device is only required to include data in the response data array up to the last implemented attribute.
4. If the length of data in the response is less than documented in the object definition and the last part of the response data does not include a complete attribute, the response data is in error and the client shall discard it.
5. If the length of data in the response is less than documented in the object definition, the client assumes default values for the missing attributes.
6. If the length of data in the response is greater than the expected amount, the client ignores any data in excess of what was expected.

**Important:** Insert default values for all unsupported attributes present in the response byte array.

**Important:** Insert a zero count and no array elements for attributes that specify a count of array elements in another attribute, if the attribute pair is not supported. An example of this type of attribute is the Discrete Input Point Object (class code 08hex), instance attribute 1, Number of Attributes.

**Important:** Insert a zero count for attributes that specify a count followed by an array of entries, if the attribute is not supported. An example of this type of attribute is class attribute 4, Optional attribute list.

Refer to Volume 1, Chapter 4 for rules for how the Get\_Attributes\_All response data is defined.

#### A-4.1.2 Request Service Data Field Parameters

None.

#### **A-4.1.3 Success Response Service Data Field Parameters**

The following information is specified within the Service Data Field of a successful response to a Get\_Attributes\_All request

**Table A-4.1 Service Data for Set\_Attributes\_All Success Response**

Name	Data Type	Description of Parameter
Attribute Data	Object/class attribute-specific Struct	A stream of information containing all of the attributes. Classes/Objects which support this service must define the format of this parameter.

## **A-4.2 Set\_Attributes\_All - Service Code: 02<sub>Hex</sub>**

Modifies the contents of the instance or class attributes defined in the object definition.

### **A-4.2.1 Service Requirements**

The following list details requirements associated with the Set\_Attributes\_All service:

1. The structure of the information in the request's Service Data Field adheres to the definition of the Set\_Attributes\_All request for the object or class. Support of this service requires the Object and/or Class to provide a detailed definition of the format of the data sent in the request message.
2. If the ability to modify an attribute changes based on the state of the object, the object definition must provide a detailed description of how this service is affected. For example, this service could only be supported in a state where all attributes are modifiable.
3. Attributes will be modified only if all attribute specific value verifications (e.g. range checks, etc.) are successful. The first attribute failing verification will be specified in the Additional Code parameter of the Error Response message.
4. If any other error is detected, then an Error Response is returned.
5. If all verification checks pass, then the attributes are modified and a Set\_Attributes\_All success response is returned.
6. If the length of data in the service is less than documented in the object definition and the last part of the response data does not include a complete attribute, the "Not enough data" (0x13) Error Response is returned.
7. If the length of data in the service data field is less than documented in the object definition, this indicates the attributes represented by the missing data are not supported by the client. Missing data shall result in no modification to the corresponding attributes.
8. If the length of data in the request is greater than the expected amount, the "Too much data" (0x15) Error Response is returned.
9. The Set\_Attributes\_All service can be supported by an object only if all settable attributes shown in the Set\_Attributes\_All request byte array are implemented and settable. The "Attribute Not Supported" (0x14) is the required error code.

Refer to Chapter 4 for rules how the Set\_Attributes\_All service data is defined.

### **A-4.2.2 Request Service Data Field Parameters**

The following information is specified within the Service Data Field of a Set\_Attributes\_All request.

**Table A-4.2 Service Data for Set\_Attributes\_All Request**

Name	Data Type	Description of Parameter
Attribute Data	Object/class attribute-specific Struct	A stream of information containing all of the attributes. Classes/Objects which support this service must define the format of this parameter.

### **A-4.2.3 Success Response Service Data Field Parameters**

None.

## **A-4.3 Get\_Attribute\_List - Service Code: 03<sub>Hex</sub>**

The Get\_Attribute\_List service shall return the contents of the selected gettable attributes of the specified object class or instance.

### **A-4.3.1 Service Requirements**

The following list details requirements associated with the Get\_Attribute\_List service:

1. The attributes shall be specified using a list of attribute identifiers.
2. The number of attributes actually returned shall be reported within the response. If there is not enough space for an attribute's data in the response message, a partial response shall be returned.
3. Attribute data returned within partial response messages shall be complete. The client application can submit another Get\_Attribute\_List service request for the attribute data remaining from its original list.
4. The client application shall verify the value of the attribute count parameter in the response.
5. Status shall be reported with each individual attribute. Attribute data shall be retrieved and packed in the sequence specified in the request.
6. When “Attribute Status” is a value other than “Success” (0x00), the “Attribute Data” shall not be returned.

### **A-4.3.2 Request Service Data Field Parameters**

The following information is specified within the Service Data Field of a Get\_Attribute\_List request.

**Table A-4.3 Service Data for Get\_Attribute\_List Request**

Name	Data Type	Description of Parameter
Attribute_count	UINT	Number of attribute identifiers in the attribute list
Attribute_list	ARRAY of UINT	List of the attribute identifiers to get from the class or object

### **A-4.3.3 Success Response Service Data Field Parameters**

The following information is specified within the Service Data Field of a successful Get\_Attribute\_List response.

**Table A-4.4 Service Data for Get\_Attribute\_List Success Response**

Name	Data Type	Description of Parameter
Attribute_count	UINT	Number of “Attribute response structures” returned
Attribute response structures	LIST of STRUCT:	A list of response structures
	UINT	“Attribute identifier”, the number of the attribute identifier being returned
	UINT	“Attribute Status”, the status of the attribute response (per appendix B)
	Object/Class specific attribute data	“Attribute Data”, Attribute response data - exists when “Attribute Status” is the value “Success” (0x00).

## **A-4.4 Set\_Attribute\_List - Service Code: 04<sub>Hex</sub>**

The Set\_Attribute\_List service shall set the contents of selected attributes of the specified object class or instance.

### **A-4.4.1 Service Requirements**

The following list details requirements associated with the Set\_Attribute\_List service:

1. The data for each individual attribute shall be placed into the request in its entirety.
2. Each set activity shall be performed in the order specified in the list.
3. Status shall be reported for each of the individual attributes in the response.
4. A server shall not attempt to set an attribute for which it can not return a response status.

### **A-4.4.2 Request Service Data Field Parameters**

The service data field of the Set\_Attribute\_List request shall be as specified in the following table.

**Table A-4.5 Service Data for Set\_Attribute\_List Request**

Name	Data Type	Description of Parameter
Attribute_count	UINT	Number of attribute identifiers in the attribute list
Attributes request structures	LIST of STRUCT:	List of structures specific to this service
	UINT	“Attribute Identifier” - Attribute identification number
	Object / class specific data	Related attribute data

### **A-4.4.3 Success Response Service Data Field Parameters**

The following information is specified within the Service Data Field of a successful Set\_Attribute\_List response.

**Table A-4.6 Service Data for Set\_Attribute\_List Response**

Name	Data Type	Description of Parameter
Attribute_count	UINT	Number of attribute values being returned
Attribute response structures	LIST of STRUCT:	A list of response structures
	UINT	“Attribute identifier”, the number of the attribute identifier being returned
	UINT	“Attribute Status”, the status of the attribute response (per appendix B)
	Object/Class specific attribute data	“Attribute Data”, Attribute response data - may exist when “Attribute Status” is the value “Success” (0x00).

## A-4.5 Reset - Service Code: 05<sub>hex</sub>

Invokes the *Reset* service of the specified Class/Object. Typically this would cause a transition to a default state or mode.

### A-4.5.1 Service Requirements

The following list details requirements associated with the Reset service:

1. If the execution of the Reset service request would place the object/device in a state that will enable it to respond to the requester, then the response is not returned until the service has completed. If the execution of the Reset service would place the object/device in a state in which it may not be able to respond, the object/device must respond prior to executing the Reset service.
2. If an error is detected, then an Error Response is returned. Otherwise, a successful Reset response is returned.

### A-4.5.2 Request Service Data Field Parameters

The following information is **OPTIONALLY** specified within the Service Data Field of a Reset request.

**Table A-4.7 Service Data for Reset Request**

Name	Data Type	Description of Parameter
Object Specific Data	Object/class service-specific STRUCT	Contains Class/Instance specific parameters. If a Class/Instance utilizes this field, then the Class/Instance definition must specify its format.

### A-4.5.3 Success Response Service Data Field Parameters

The following information is **OPTIONALLY** specified within the Service Data Field of a successful response to a Reset request

**Table A-4.8 Service Data for Reset Success Response**

Name	Data Type	Description of Parameter
Object Specific Data	Object/class service-specific STRUCT	Contains Class/Instance specific parameters. If a Class/Instance utilizes this field, then the Class/Instance definition must specify its format.

## **A-4.6 Start - Service Code: 06<sub>Hex</sub>**

Invokes the *Start* service of the specified Class/Object. Typically, this would place an object into a running state/mode.

### **A-4.6.1 Service Requirements**

The following list details requirements associated with the Start service:

1. Other than documenting the state machine associated with the object/class relative to this service, there are no special requirements defined by CIP.
2. If an error is detected, then an Error Response is returned. Otherwise, a successful Start response is returned.

### **A-4.6.2 Request Service Data Field Parameters**

The following information is **OPTIONALLY** specified within the Service Data Field of a Start request.

**Table A-4.9 Service Data for Start Request**

Name	Data Type	Description of Parameter
Object Specific Data	Object/class service-specific STRUCT	Contains Class/Instance specific parameters. If a Class/Instance utilizes this field, then the Class/Instance definition must specify its format.

### **A-4.6.3 Success Response Service Data Field Parameters**

The following information is **OPTIONALLY** specified within the Service Data Field of a successful response to a Start request

**Table A-4.10 Service Data for Start Success Response**

Name	Data Type	Description of Parameter
Object Specific Data	Object/class service-specific STRUCT	Contains Class/Instance specific parameters. If a Class/Instance utilizes this field, then the Class/Instance definition must specify its format.

## **A-4.7 Stop - Service Code: 07<sub>Hex</sub>**

Invokes the *Stop* service of the specified Class/Object. Typically, this would place an object into a stopped or idle state/mode.

### **A-4.7.1 Service Requirements**

The following list details requirements associated with the Stop service:

1. Other than documenting the state machine associated with the object/class relative to this service, there are no special requirements defined by CIP.
2. If an error is detected, then an Error Response is returned. Otherwise, a successful Stop response is returned.

### **A-4.7.2 Request Service Data Field Parameters**

The following information is **OPTIONALLY** specified within the Service Data Field of a Stop request.

**Table A-4.11 Service Data for Stop Request**

Name	Data Type	Description of Parameter
Object Specific Data	Object/class service-specific STRUCT	Contains Class/Instance specific parameters. If a Class/Instance utilizes this field, then the Class/Instance definition must specify its format.

### **A-4.7.3 Success Response Service Data Field Parameters**

The following information is **OPTIONALLY** specified within the Service Data Field of a successful response to a Stop request

**Table A-4.12 Service Data for Stop Success Response**

Name	Data Type	Description of Parameter
Object Specific Data	Object/class service-specific STRUCT	Contains Class/Instance specific parameters. If a Class/Instance utilizes this field, then the Class/Instance definition must specify its format.

## A-4.8 Create - Service Code: 08<sub>Hex</sub>

Results in the instantiation of a new object within the specified class.

### A-4.8.1 Service Requirements

The following list details requirements associated with the Create service:

1. The object instance is created and initialized in accordance with the object definition.
2. Any error will result in the object instance not being created.
3. If an error is detected, then an Error Response is returned. Otherwise, a successful Create response is returned.

### A-4.8.2 Request Service Data Field Parameters

The following information is **OPTIONALLY** specified within the Service Data Field of a Create request.

**Table A-4.13 Service Data for Create Request**

Name	Data Type	Description of Parameter
Object Specific Data	Object/class service-specific STRUCT	Contains Class/Instance specific parameters. If a Class/Instance utilizes this field, then the Class/Instance definition must specify its format.

### A-4.8.3 Success Response Service Data Field Parameters

The following information is specified within the Service Data Field of a successful response to a Create request

**Table A-4.14 Service Data for Create Success Response**

Name	Data Type	Description of Parameter
Instance ID	UINT	The integer value assigned to identify the newly created object. This is specified within a 16-bit field regardless of the Message Body Format associated with the Explicit Messaging Connection.
Object Specific Data	Object/class service-specific STRUCT	Contains Class/Instance specific parameters. If a Class/Instance utilizes this field, then the Class/Instance definition must specify its format.

## **A-4.9 Delete - Service Code: 09<sub>Hex</sub>**

Deletes an object instance of the specified class.

### **A-4.9.1 Service Requirements**

The following list details requirements associated with the Delete service:

1. All resources are deallocated and returned to the system.
2. If an error is detected, then an Error Response is returned. Otherwise, a successful Delete response is returned.

### **A-4.9.2 Request Service Data Field Parameters**

The following information is **OPTIONALLY** specified within the Service Data Field of a Delete request.

**Table A-4.15 Service Data for Delete Request**

Name	Data Type	Description of Parameter
Object Specific Data	Object/class service-specific STRUCT	Contains Class/Instance specific parameters. If a Class/Instance utilizes this field, then the Class/Instance definition must specify its format.

### **A-4.9.3 Success Response Service Data Field Parameters**

The following information is **OPTIONALLY** specified within the Service Data Field of a successful response to a Delete request

**Table A-4.16 Service Data for Delete Success Response**

Name	Data Type	Description of Parameter
Object Specific Data	Object/class service-specific STRUCT	Contains Class/Instance specific parameters. If a Class/Instance utilizes this field, then the Class/Instance definition must specify its format.

## **A-4.10    Multiple\_Service\_Packet - Service Code: 0A<sub>Hex</sub>**

Performs a set of services as an autonomous sequence.

### **A-4.10.1    Service Requirements**

The following list details requirements associated with the Multiple\_Service\_Packet service:

1. Performs services as an autonomous sequence of services.
2. Performs services in the sequence supplied.
3. Performs all services, reporting individual responses for each one.
4. Packs responses into the response buffer in the sequence in which they were executed.
5. A response timeout must be implemented for those service requests that do not guarantee a response.
6. Each embedded service may return a success or failure, as indicated in the response structure. If one or more service requests result in an error this service shall return an error. The error code returned shall be 1E<sub>hex</sub> (Embedded service error).

This service allows clients to submit a sequence of ‘embedded’ services in a single message packet. The object processing the Multiple\_Service\_Packet shall not perform any other service until the entire sequence of embedded services has been attempted.

The embedded services are formatted according to their definitions. Each embedded service may contain an EPATH. If an EPATH is present, the embedded service is passed to the Message Router for processing. If an EPATH is not present, the object performing the Multiple\_Service\_Packet request shall perform the embedded service.

### **A-4.10.2    Request Service Data Field Parameters**

The following information is specified within the Service Data Field of a Multiple\_Service\_Packet request.

**Table A-4.17 Service Data for Multiple\_Service\_Packet Request**

Name	Data Type	Description of Parameter
Number of Services	UINT	Number of embedded services in the Service List.
Service Offsets	ARRAY of UINT	Array of byte offsets to the start of each embedded service in the Service List.
Service List	ARRAY of STRUCT of	Array of service request structures. The format of the service request follows the Message Router Request header defined in Chapter 2.
	USINT	Service code of the request.
	USINT	The number of 16 bit words in the Request_Path field (next element).
	Padded EPATH	This is an array of bytes whose contents convey the path of the request (Class ID, Instance ID, etc.) for this transaction.
	ARRAY of octet	Service specific data to be delivered in the Explicit Messaging Request. If no additional data is to be sent with the Explicit Messaging Request, then this array will be empty.

### **A-4.10.3 Success Response Service Data Field Parameters**

The following information is specified within the Service Data Field of a successful response to a Multiple\_Service\_Packet request

**Table A-4.18 Service Data for Multiple\_Service\_Packet Response**

Name	Data Type	Description of Parameter
Number of Responses	UINT	Number of embedded services responses in the Response List.
Response Offsets	ARRAY of UINT	Array of byte offsets to the start of each embedded service response in the Response List.
Response List	ARRAY of STRUCT of	Array of service response structures. The format of the service response follows the Message Router Response header defined in Chapter 2.
	USINT	Reply service code.
	USINT	Shall be zero.
	USINT	One of the General Status codes listed in Appendix B (Status Codes).
	USINT	Number of 16 bit words in Additional Status array.
	ARRAY of UINT	Additional status.
	ARRAY of octet	Response data from request or additional error data if General Status indicated an error.

## **A-4.11 Apply\_Attributes - Service Code: 0D<sub>Hex</sub>**

Causes attribute values whose use is *pending* to become actively used.

### **A-4.11.1 Service Requirements**

The following list details requirements associated with the Apply\_Attributes service:

1. Data for pending attributes must be verified before it is actually used.
2. If an error is detected, then an Error Response is returned. Otherwise, a successful Apply\_Attributes response is returned.

### **A-4.11.2 Request Service Data Field Parameters**

The following information is **OPTIONALLY** specified within the Service Data Field of an Apply\_Attributes request.

**Table A-4.19 Service Data for Apply\_Attributes Request**

Name	Data Type	Description of Parameter
Object Specific Data	Object/class service-specific STRUCT	Contains Class/Instance specific parameters. If a Class/Instance utilizes this field, then the Class/Instance definition must specify its format.

### **A-4.11.3 Success Response Service Data Field Parameters**

The following information is **OPTIONALLY** specified within the Service Data Field of a successful response to an Apply\_Attributes request

**Table A-4.20 Service Data for Apply\_Attributes Success Response**

Name	Data Type	Description of Parameter
Object Specific Data	Object/class service-specific STRUCT	Contains Class/Instance specific parameters. If a Class/Instance utilizes this field, then the Class/Instance definition must specify its format.

## **A-4.12 Get\_Attribute\_Single - Service Code: 0E<sub>Hex</sub>**

Returns the contents of the specified attribute.

### **A-4.12.1 Service Requirements**

The following list details requirements associated with the Get\_Attribute\_Single service:

1. The service causes the class/object to return the contents of the specified attribute to the requester.
2. If an error is detected, then an Error Response is returned. Otherwise, a successful Get\_Attribute\_Single response is returned along with the requested attribute data.

### **A-4.12.2 Request Service Data Field Parameters**

The following information is specified within the Service Data Field of a Get\_Attribute\_Single request.

**Table A-4.21 Service Data for Get\_Attribute\_Single Request**

Name	Data Type	Description of Parameter
Attribute ID <sup>1</sup>	USINT	Identifies the attribute to be read/returned

<sup>1</sup> This parameter shall be present when the target of the request is on a subnet, which does not support explicit message addressing to the attribute level. If the subnet does support attribute level addressing then this parameter shall not be present. In the latter case, the data type of the Attribute ID can be either USINT, UINT, or UDINT.

### **A-4.12.3 Success Response Service Data Field Parameters**

The following information is specified within the Service Data Field of a successful Get\_Attribute\_Single response.

**Table A-4.22 Service Data for Get\_Attribute\_Single Success Response**

Name	Data Type	Description of Parameter
Attribute Data	Object/class attribute –specific Struct	Contains the requested attribute data

## **A-4.13 Set\_Attribute\_Single - Service Code: 10<sub>Hex</sub>**

Modifies an attribute value.

### **A-4.13.1 Service Requirements**

The following list details requirements associated with the Set\_Attribute\_Single service:

1. The attribute data is validated prior to the modification taking place.
2. If an error is detected, then an Error Response is returned. Otherwise a successful Set\_Attribute\_Single response is returned.

### **A-4.13.2 Request Service Data Field Parameters**

The following information is specified within the Service Data Field of a Set\_Attribute\_Single request.

**Table A-4.23 Service Data for Set\_Attribute\_Single Request**

Name	Data Type	Description of Parameter
Attribute ID <sup>1</sup>	USINT	Identifies the attribute to be read/returned
Attribute Data	Attribute specific	Contains the value to which the specified attribute is to be modified.

<sup>1</sup> This parameter shall be present when the target of the request is on a subnet, which does not support explicit message addressing to the attribute level. If the subnet does support attribute level addressing then this parameter shall not be present. In the latter case, the data type of the Attribute ID can be either USINT, UINT, or UDINT.

### **A-4.13.3 Success Response Service Data Field Parameters**

The following information is OPTIONALLY specified within the Service Data Field of a successful response to a Set\_Attribute\_Single request

**Table A-4.24 Service Data for Set\_Attribute\_Single Success Response**

Name	Data Type	Description of Parameter
Object Data	Object/class service-specific STRUCT	Contains Class/Instance specific parameters. If a Class/Instance utilizes this field, then the Class/Instance definition must specify its format.

## **A-4.14 Find\_Next\_Object\_Instance - Service Code: 11<sub>Hex</sub>**

This service is supported at the Class level only. It causes the specified Class to search for and return a list of Instance IDs associated with existing Object Instances. *Existing* Objects are those that are currently accessible from the CIP subnet.

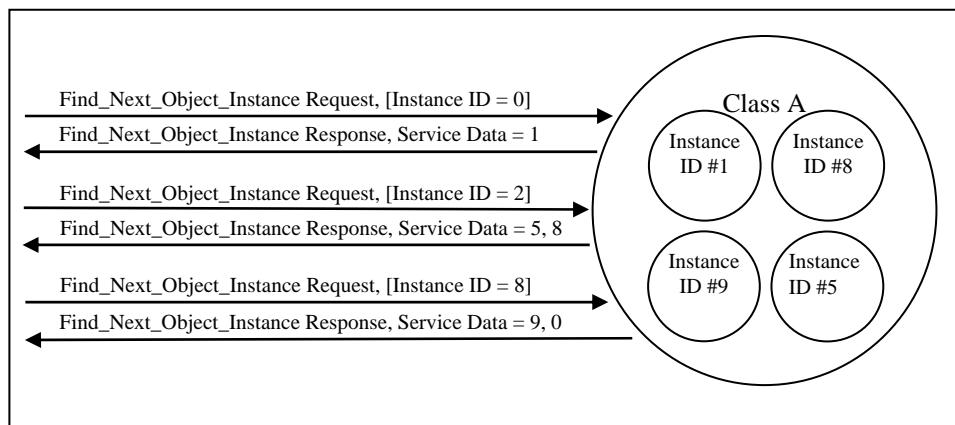
### **A-4.14.1 Service Requirements**

The following list details requirements associated with the Find\_Next\_Object\_Instance service:

1. The Class utilizes the value specified in the Instance ID of the request message to determine the starting point for the search as described below:
  - If the Instance ID in the request message is zero (0), then the Class starts with the numerically lowest Instance ID.
  - If the Instance ID in the request message is not zero (0), then the Class starts with the next Instance ID whose value is numerically greater than the specified Instance ID.
  - If the Instance ID in the request message is greater than or equal to the numerically highest Instance ID within the Class, then the value zero (0) is returned.
2. The Class returns a list of Instance IDs associated with existing Objects beginning at the starting point and continuing in ascending Instance ID value order.
3. The request specifies the maximum number of Instance ID values to be returned in the response. The responding Class can return any number of Instance IDs less than or equal to the maximum specified in the request.
4. The responding device returns Instance ID value zero (0) to indicate that the end of the list has been reached.
5. If an error is detected, then an Error Response is returned. Otherwise, a successful Find\_Next\_Object\_Instance response is returned.

The following illustration provides a general example of this service. Specific encoding examples are presented in section A-6.

**Figure A-4.25 Example of Find\_Next\_Object\_Instance Message Flow**



#### **A-4.14.2 Request Service Data Field Parameters**

The following information is specified within the Service Data Field of a Find\_Next\_Object\_Instance request.

**Table A-4.26 Service Data for Find\_Next\_Object\_Instance Request**

Name	Data Type	Description of Parameter
Maximum Returned Values	USINT	Indicates the <b>maximum</b> number of Instance ID values to be returned in the response message.

#### **A-4.14.3 Success Response Service Data Field Parameters**

The following information is specified within the Service Data Field of a successful Find\_Next\_Object\_Instance response.

**Table A-4.27 Service Data for Find\_Next\_Object\_Instance Success Response**

Name	Data Type	Description of Parameter
Number Of List Members	USINT	Contains the number of Instance IDs specified in this response message.
Instance ID List	Array of UINT	Contains the returned Instance ID List. The Instance IDs are returned in 16 bit integer fields.

## **A-4.15 Restore - Service Code: 15<sub>Hex</sub>**

Restores the contents of a class/object's attributes from a storage location accessible by the Save service. Attribute data is copied from a storage area to the *currently active* memory area used by the class/object.

### **A-4.15.1 Service Requirements**

The following list details requirements associated with the Restore service:

1. Attribute data must be verified before the copy from the “storage area” to the “actively used” area is performed.
2. If the ability to modify an attribute changes based on the state of the object, the object definition must provide a detailed description of how this service is effected. For example; this service could only be supported in a state where all attributes are modifiable. Alternatively, the object could ignore the data associated with a currently non-modifiable attribute.
3. Attributes will be modified only if all attribute specific value verifications (e.g. range checks, etc.) are successful. The first attribute failing verification will be specified in the Additional Code parameter of the Error Response message.
4. If any other error is detected, then an Error Response is returned.
5. If all verification checks pass, then the attributes are modified and a Restore success response is returned.

### **A-4.15.2 Request Service Data Field Parameters**

The following information is **OPTIONALLY** specified within the Service Data Field of a Restore request.

**Table A-4.28 Service Data for Restore Request**

Name	Data Type	Description of Parameter
Object Specific Data	Object/class service-specific STRUCT	Contains Class/Instance specific parameters. If a Class/Instance utilizes this field, then the Class/Instance definition must specify its format.

### **A-4.15.3 Success Response Service Data Field Parameters**

The following information is **OPTIONALLY** specified within the Service Data Field of a successful response to a Restore request

**Table A-4.29 Service Data for Restore Success Response**

Name	Data Type	Description of Parameter
Object Specific Data	Object/class service-specific STRUCT	Contains Class/Instance specific parameters. If a Class/Instance utilizes this field, then the Class/Instance definition must specify its format.

## **A-4.16 Save - Service Code: 16<sub>Hex</sub>**

Copies the contents of an class/object's attributes to a location accessible by the Restore service.

### **A-4.16.1 Service Requirements**

The following list details requirements associated with the Save service:

1. The service will report success only after the copy has been completed and verified.
2. If an error is detected, then an Error Response is returned. Otherwise, a successful Save response is returned.

### **A-4.16.2 Request Service Data Field Parameters**

The following information is **OPTIONALLY** specified within the Service Data Field of a Save request.

**Table A-4.30 Service Data for Save Request**

Name	Data Type	Description of Parameter
Object Specific Data	Object/class service-specific STRUCT	Contains Class/Instance specific parameters. If a Class/Instance utilizes this field, then the Class/Instance definition must specify its format.

### **A-4.16.3 Success Response Service Data Field Parameters**

The following information is **OPTIONALLY** specified within the Service Data Field of a successful response to a Save request

**Table A-4.31 Service Data for Save Success Response**

Name	Data Type	Description of Parameter
Object Specific Data	Object/class service-specific STRUCT	Contains Class/Instance specific parameters. If a Class/Instance utilizes this field, then the Class/Instance definition must specify its format.

## **A-4.17 No Operation (NOP) - Service Code: 17<sub>Hex</sub>**

This service merely causes the receiving object to return a *No Operation* response. The receiving object does not carry out any other internal action. This service can be used to test whether or not a particular object is still present and responding without causing a state change.

### **A-4.17.1 Required Behavior**

The NOP service requires the following behaviors:

1. If the object to which the request is delivered supports the service, then a response whose status indicates success is returned. If the object does not support the service, then a response indicating an error was detected is returned.

### **A-4.17.2 Request Service Data Field Parameters**

NONE

### **A-4.17.3 Success Response Service Data Field Parameters**

NONE

## **A-4.18 Get\_Member - Service Code: 18<sub>Hex</sub>**

Returns member(s) information from within an attribute. See Section A-5 for *Member Service Protocol* details.

### **A-4.18.1 Service Requirements**

The following list details requirements associated with the Get\_Member service:

1. The service causes the class/object to return member(s) at the specified Member ID of an attribute.
2. If an error is detected, then an Error Response is returned. Otherwise, a successful Get\_Member response is returned.
3. If *Member ID* is greater than largest existing *Member ID*, gets the one member with the highest Member ID.
4. If the *Member ID* value is zero, returns an *Invalid Member ID* Error Response. To get all members, use a Get\_Attribute\_Single service.

### **A-4.18.2 Request Service Data Field Parameters**

**Table A-4.32 Service Data for Get\_Member Request**

Request Field	Required/Optional/Not Present
Member Data	Not present

### **A-4.18.3 Success Response Service Data Field Parameters**

**Table A-4.33 Service Data for Get\_Member Response**

Response Field	Required/Optional/Not Present
Member ID	Required
Member Data	Required

## **A-4.19 Set\_Member - Service Code: 19<sub>Hex</sub>**

Sets member(s) information in an attribute. See Section A-5 for *Member Service Protocol* details.

### **A-4.19.1 Service Requirements**

The following list details requirements associated with the Set\_Member service:

1. The service causes the class/object to set member(s) at the specified Member ID of an attribute.
2. The request is checked, and if valid, the member(s) are set to the new Member Data.
3. If an error is detected, then an Error Response is returned. Otherwise, a successful Set\_Member response is returned.
4. If the Member ID value is greater than that of the last Member ID for the specific attribute, no action occurs and an Invalid Member ID Error Response is returned.
5. If multiple members are specified and fewer members exist at the specified Member ID, no action occurs and an Invalid Member ID Error Response is returned.
6. If the Member ID value is zero, an Invalid Member ID Error Response is returned.

### **A-4.19.2 Request Service Data Field Parameters**

**Table A-4.34 Service Data for Set\_Member Request**

Request Field	Required/Optional/Not Present
Member Data	Required

### **A-4.19.3 Success Response Service Data Field Parameters**

**Table A-4.35 Service Data for Set\_Member Response**

Response Field	Required/Optional/Not Present
Member ID	Conditional (Required if Member Data is present)
Member Data	Optional

## **A-4.20 Insert\_Member - Service Code: 1A<sub>Hex</sub>**

Inserts member(s) into an attribute. See Section A-5 for *Member Service Protocol* details.

### **A-4.20.1 Service Requirements**

The following list details requirements associated with the Insert\_Member service:

1. The service causes the class/object to insert member(s) at the specified *Member ID* of an attribute. The Member IDs of members at or following the specified Member ID will change.
2. The request is checked, and if valid, the member(s) are inserted. If member data is not included in the request, the member(s) are set to the class/attribute specific default.
3. If an error is detected, then no action is taken and an Error Response is returned. Otherwise, a successful Insert\_Member response is returned.
4. If the number of members specified cannot be added to the attribute, then a “Resource unavailable” error response is returned.
5. If the Member ID value is greater than that of the last *Member ID* for the specific attribute, appends member(s) to the attribute and returns the *Member ID* where inserted.
6. If the *Member ID* value is zero, returns an *Invalid Member ID* Error Response.

### **A-4.20.2 Request Service Data Field Parameters**

**Table A-4.36 Service Data for Insert\_Member Request**

Request Field	Required/Optional/Not Present
Member Data	Optional

### **A-4.20.3 Success Response Service Data Field Parameters**

**Table A-4.37 Service Data for Insert\_Member Request**

Response Field	Required/Optional/Not Present
Member ID	Required
Member Data	Optional

## **A-4.21 Remove\_Member - Service Code: 1B<sub>Hex</sub>**

Removes member(s) from an attribute. See Section A-4.22 for *Member Service Protocol* details.

### **A-4.21.1 Service Requirements**

The following list details requirements associated with the Remove\_Member service:

1. The service causes the class/object to remove member(s) at the specified *Member ID* of an attribute. The Member IDs of members following the removed member(s) change.
2. If an error is detected, then no action is taken and an Error Response is returned. Otherwise, a successful Remove\_Member response is returned.
3. If *Member ID* is greater than largest existing *Member ID*, the one member with the highest Member ID is removed.
4. If multiple members are specified and fewer members exist at the specified Member ID, the member(s) that do exist are removed.

### **A-4.21.2 Request Service Data Field Parameters**

**Table A-4.38 Service Data for Remove\_Member Request**

Request Field	Required/Optional/Not Present
Member Data	Not present

### **A-4.21.3 Success Response Service Data Field Parameters**

**Table A-4.39 Service Data for Remove\_Member Response**

Response Field	Required/Optional/Not Present
Member ID	Required
Member Data	Required

## **A-4.22 GroupSync - Service Code: 1C<sub>Hex</sub>**

Verify each member of a group is synchronized to System Time.

### **A-4.22.1 Service Requirements**

The following list details requirements associated with the GroupSync service:

1. The structure of the information in the request's Service Data Field adheres to the definition of the GroupSync request for the object or class. Support of this service requires the Object and/or Class to provide a detailed definition of the format of the data sent in the request message.
2. The structure of the information in the response's Service Data Field adheres to the definition of the GroupSync response for the object or class. Support of this service requires the Object and/or Class to provide a detailed definition of the format of the data sent in the response message.
3. The responder will verify that the application is synchronized according to the object specific requirements. The target return's a 1 in the IsSynchronized attribute of the response Service Data Field if the synchronized status is true and a 0 if the synchronized status is false.

Refer to the GroupSync section of the CIP Sync Overview in the Time Sync Object for a more detailed description of the GroupSync service.

### **A-4.22.2 Request Service Data Field Parameters**

The following information is specified within the Service Data Field of a GroupSync request.

**Table A-4.40 Service Data for GroupSync Request**

Name	Data Type	Description of Parameter
Object specific parameters	Object specific	Object specific

### **A-4.22.3 Response Service Data Field Parameters**

The following information is specified within the Service Data Field of a GroupSync response.

**Table A-4.41 Service Data for GroupSync Success Response**

Name	Data Type	Description of Parameter	Semantics of values
IsSynchronized	BOOL	Indicates if object is synchronized to the PTP Time Master	1 = Is Synchronized 0 = Not Synchronized
Object specific parameters	Object specific	Object specific	

## A-5 Member Service Protocols

Attributes of object classes may consist of arrays of basic data types or structures. To manipulate members of an array, the following member services use the protocols defined in this section:

- Get\_Member
- Set\_Member
- Insert\_Member
- Remove\_Member

The first member in an array is specified by a *Member ID* value of one (1).

### A-5.1 Member ID/EX Description

Two message formats are defined for member services: basic and extended. Within the extended format, there exist up to 255 protocol options. The message format is selected by the most significant bit of the Member ID. When a subnet does not support Attribute and Member level addressing, the Member ID/EX parameter is sent as a 16 bit (WORD) parameter within the service data. If the subnet does support Attribute and Member level addressing, the Member ID/EX parameter is sent within the Logical Segment and can be either 8, 16, or 32 bits (BYTE, WORD, DWORD). The following table defines the bits contained within the *Member ID/EX* field when sent as a WORD.

**Table A-5.1 Member ID/EX Description (WORD)**

Bits							
7	6	5	4	3	2	1	0
Member ID Bits 0-7							
EX	Member ID Bits 8-14						

The lower fifteen bits (0-14) of the *Member ID/EX* field identify the member to be manipulated. Bit 15 (EX), selects either the basic (EX=0) or extended (EX=1) message format.

## A-5.2 Member Request Message - Basic

The following table illustrates the basic *Member Request Message* service data format.

**Table A-5.2 Service Data for Basic Format Member Request Messages**

Name	Data Type	Description of Parameter
Attribute ID <sup>1</sup>	USINT	Identifies the attribute the member is to be serviced.
Member ID/ EX <sup>2</sup>	WORD	The lower 15 bits identifies the Member ID identifies attribute of the new member to be serviced. If the value is zero (0) the service used determines the behavior.  Extended Protocol Option EX = 0 (Bit 15)
Member Data (conditional)	Member specific	This field is required by some services, see individual service descriptions.

<sup>1</sup> This parameter shall be present when the target of the request is on a subnet which does not support explicit message addressing to the attribute level. If the subnet does support attribute level addressing then this parameter shall not be present. In the latter case, the data type of the Attribute ID can be either USINT, UINT, or UDINT.

<sup>2</sup> This parameter shall be present when the target of the request is on a subnet, which does not support explicit message addressing to the member or attribute level. If the subnet does support attribute and member level addressing then this parameter shall not be present. In this latter case, the data type of the Member ID/EX can be either BYTE, WORD, or DWORD with the most significant bit being the Extended Protocol Option parameter and the remaining bits being the Member ID.

**Table A-5.3 Service Data for Basic Format Member Response Messages**

Name	Data Type	Description of Parameter
Member ID (conditional)	UINT	The Member ID of the member serviced.  This field is required by some services, see individual service descriptions. This field is required if the Member Data field is present.
Member Data (conditional)	Member specific	This field is required by some services, see individual service descriptions.

### A-5.3 Member Request Message - Extended

If the Extended protocol bit [EX] of the *Member ID/EX* field is set to a one [1], the byte following the *Member ID/EX* defines the remainder of the protocol.

The *Extended Protocol ID* value conforms to the standard reserved ranges for open/vendor specific/ reserved ranges.

**Table A-5.4 Service Data for Extended Format Member Request Messages**

Name	Data Type	Description of Parameter
Attribute ID <sup>1</sup>	USINT	Identifies the attribute the member is to be serviced.
Member ID / EX <sup>2</sup>	WORD	The lower 15 bits identifies the Member ID identifies attribute of the new member to be serviced. If the value is zero (0) the service used determines the behavior. Extended Protocol Option EX= 1 (Bit 15)
Extended Protocol ID	USINT	Selects the extended protocol used for the remainder of message, per Extended Protocol ID table.
	Protocol specific	Additional request data is defined by the extended protocol.

<sup>1</sup> This parameter shall be present when the target of the request is on a subnet, which does not support explicit message addressing to the attribute level. If the subnet does support attribute level addressing then this parameter shall not be present. In the latter case, the data type of the Attribute ID can be either USINT, UINT, or UDINT.

<sup>2</sup> This parameter shall be present when the target of the request is on a subnet, which does not support explicit message addressing to the member or attribute level. If the subnet does support attribute and member level addressing then this parameter shall not be present. In this latter case, the data type of the Member ID/EX can be either BYTE, WORD, or DWORD with the most significant bit being the Extended Protocol Option parameter and the remaining bits being the Member ID.

**Table A-5.5 Service Data for Extended Format Member Response Messages**

Name	Data Type	Description of Parameter
	Protocol specific	Response data is defined by the extended protocol option.

The following table contains a list of the currently defined values for the Extended Protocol ID.

**Table A-5.6 Extended Protocol ID**

Value	Description
0	Reserved for future CIP use
1	Multiple Sequential Members
2	International String Selection
3-63hex	Reserved for future CIP use
64hex-C7hex	Vendor Specific
C8hex-FFhex	Reserved for future CIP use

### A-5.3.1 Multiple Sequential Members - Extended Protocol

When the *Extended Protocol ID* is set to *Multiple Sequential Members* [1], the following protocol is used:

**Table A-5.7 Service Data for Multiple Sequential Member Request Messages**

Name	Data Type	Description of Parameter
Attribute ID <sup>1</sup>	USINT	Identifies the attribute the member is to be serviced.
Member ID / EX <sup>2</sup>	WORD	The lower 15 bits identifies the Member ID identifies attribute of the new member to be serviced. If the value is zero (0) the service used determines the behavior. Extended Protocol Option EX= 1 (Bit 15)
Extended Protocol ID	USINT [01]	Selects the Multiple Sequential Members protocol option
Number of Members	UINT	The number of members to be serviced.
Member Data (conditional)	Member specific	Multiple sequential Member Data This field is required by some services, see individual service descriptions.

<sup>1</sup> This parameter shall be present when the target of the request is on a subnet, which does not support explicit message addressing to the attribute level. If the subnet does support attribute level addressing then this parameter shall not be present. In this latter case, the data type of the Attribute ID can be either USINT, UINT, or UDINT.

<sup>2</sup> This parameter shall be present when the target of the request is on a subnet, which does not support explicit message addressing to the member or attribute level. If the subnet does support attribute and member level addressing then this parameter shall not be present. In the latter case, the data type of the Member ID/EX can be either BYTE, WORD, or DWORD with the most significant bit being the Extended Protocol Option parameter and the remaining bits being the Member ID.

**Table A-5.8 Service Data for Member Response Messages (Extended Protocol)**

Name	Data Type	Description of Parameter
Number of Members	UINT	The number of members serviced. This field is required.
Member ID (conditional)	UINT	The Member ID of the first member serviced. This field is required by some services, see individual service descriptions. This field is required if the Member Data field is present.
Member Data (conditional)	Member specific	Multiple sequential Member Data This field is required by some services, see individual service descriptions.

### A-5.3.2 International String Selection– Extended Protocol

Returns a single international character string from an attribute, which has a data type of STRINGI. This service can request a specific language or default to the current ‘active’ language as specified by the Identity Object.

#### Service Requirements

The International String Selection protocol requires the following behavior:

1. This member protocol causes the class/object to return an international string from an attribute of data type STRINGI.
2. This member protocol is only valid with the Get\_Member service.
3. If the object *or* attribute to which the request is delivered does not support the service/protocol, then a Service Not Supported error is returned.
4. If the attribute does not contain the requested language, then an Invalid Parameter error is returned.
5. If the Member ID parameter is zero, then the device shall return the active (default) language. This default language is set by the device and may optionally be changed through the Active Language attribute (Attribute ID 11) within the Identity object.
6. A device shall support the Supported Language List attribute (Attribute ID 12) of the Identity object if this member protocol is supported.
7. If the requested Member ID is valid but the string for that language is not currently available then an error shall be returned indicating Resource Unavailable (error code 0x02) and no additional error code. The string for the active language (as indicated by the Active Language attribute of the Identity Object) shall always be available.

**Table A-5.9 Service Data for International String Selection Request Messages**

Name	Data Type	Description of Parameter
Attribute ID	USINT	Identifies the attribute containing the international string.
Member ID / EX	WORD	The lower 15 bits identifies the language to retrieve. This value is the member ID (array index, starting at one) of the language within the Supported Language List attribute (Attribute ID 12) of the Identity object.  Extended Protocol Option EX = 1 (Bit 15)
Extended Protocol ID	USINT	Selects the International String Selection protocol option (value = 2).

**Table A-5.10 Service Data for International String Selection Response Messages**

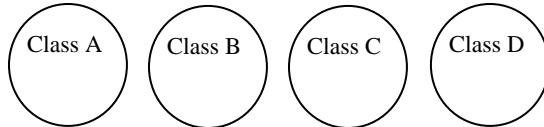
Name	Data Type	Description of Parameter
International String	STRUCT of	The character string in the requested language. This parameter follows the definition of a single international character string within the STRINGI data type (CIP Common, Appendix C-4 Abstract Syntax Specification).
	USINT	The language1 field from the STRINGI data type.
	USINT	The language2 field from the STRINGI data type.
	USINT	The language3 field from the STRINGI data type.
	EPAUTH	Encoded Data Type of the international string.
	UINT	Character set of the international string.
	OCTET_STRING	International string

## A-6 CIP Encoding Examples

The example object definitions presented in the following section are used as the basis for providing CIP Common Service encoding examples.

### A-6.1 Example Object Class Definitions

Table A-6.1 lists the class codes of the example Object Classes.



**Table A-6.1 Class ID Codes For Example Classes**

Class Name	Class ID Code
Class A	70 <sub>hex</sub>
Class B	71 <sub>hex</sub>
Class C	72 <sub>hex</sub>
Class D	73 <sub>hex</sub>

The tables below illustrate the attributes associated with Object Instances within these classes.

**Table A-6.2 Class A Object Instance Attributes**

Attribute Name	Attribute ID	Format	Access Method	Attribute Description
attribute_1	1	UINT	Gettable/ Settable	Allows values in range of 1 - 9
attribute_2	2	SINT	Gettable	No limit to value
attribute_3	3	DINT	Gettable/ Settable	No limit to value

**Table A-6.3 Class B Object Instance Attributes**

Attribute Name	Attribute ID	Format	Access Method	Attribute Description
attribute_1	1	DINT	Gettable/ Settable	No limit to value
attribute_2	2	SINT	Gettable	No limit to value
attribute_3	3	UINT	Gettable/ Settable	Value limited to range of 10 - 20
attribute_4	4	UINT	Gettable/ Settable	Value limited to range of 0 - 9

**Table A-6.4 Class C Object Instance Attributes**

Attribute Name	Attribute ID	Format	Access Method	Attribute Description
attribute_1	1	UINT	Gettable	No limit to value
attribute_2	2	UINT	Gettable	No limit to value
attribute_3	3	UINT	Gettable/ Settable	Allows values in range of 1 - 5

**Table A-6.5 Class D Object Instance Attributes**

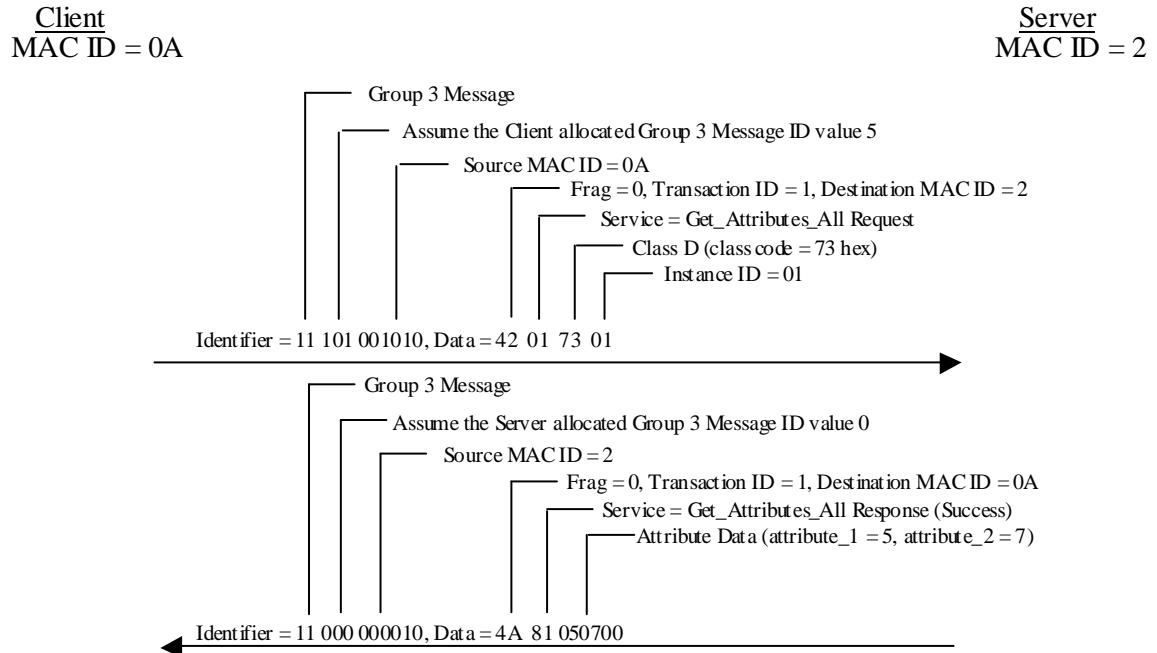
<b>Attribute Name</b>	<b>Attribute ID</b>	<b>Format</b>	<b>Access Method</b>	<b>Attribute Description</b>
attribute_1	1	USINT	Gettable/ Settable	No limit to value
attribute_2	2	UINT	Gettable/ Settable	No limit to value

## **A-6.2 Encoding Examples**

Examples showing how the CIP Common Services are encoded are provided on the following pages. Note that the examples assume that an Explicit Messaging Connection has already been established and, as such, the Connection IDs associated with the Explicit Messaging Connections have already been agreed upon.

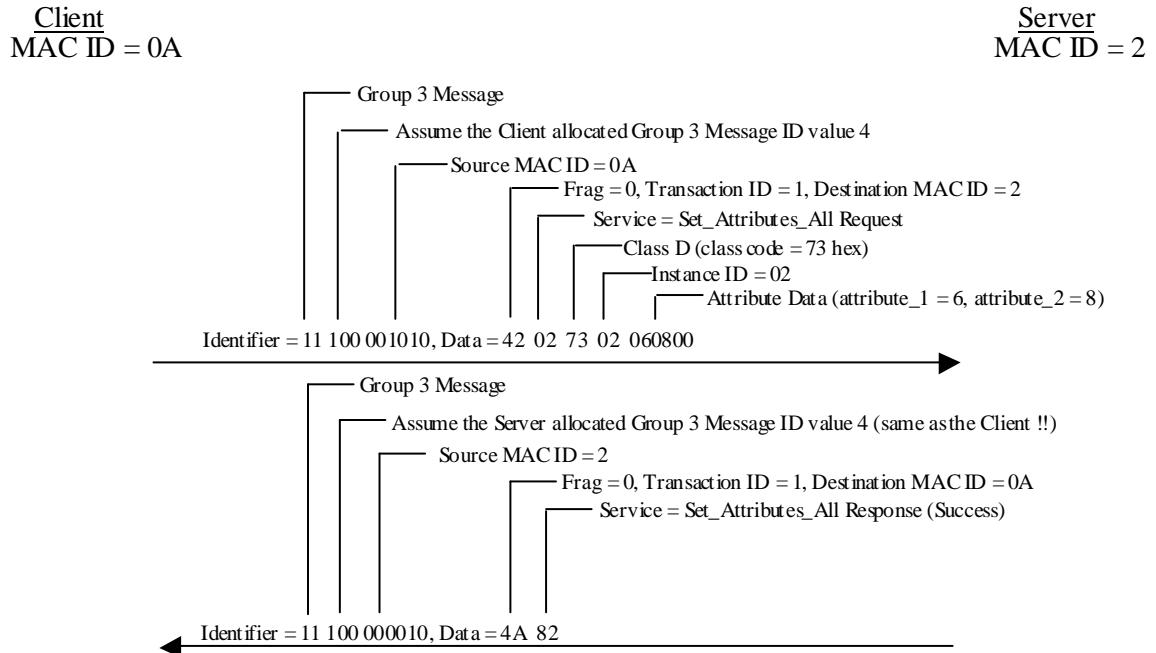
### A-6.2.1 Get\_Attributes\_All

The Client wants to read all of the attributes from Instance #1 within Class D with one request. Assume the Explicit Messaging Connection was established across Group 3 and the Message Body Format was defined by the Server as DeviceNet (8/8).



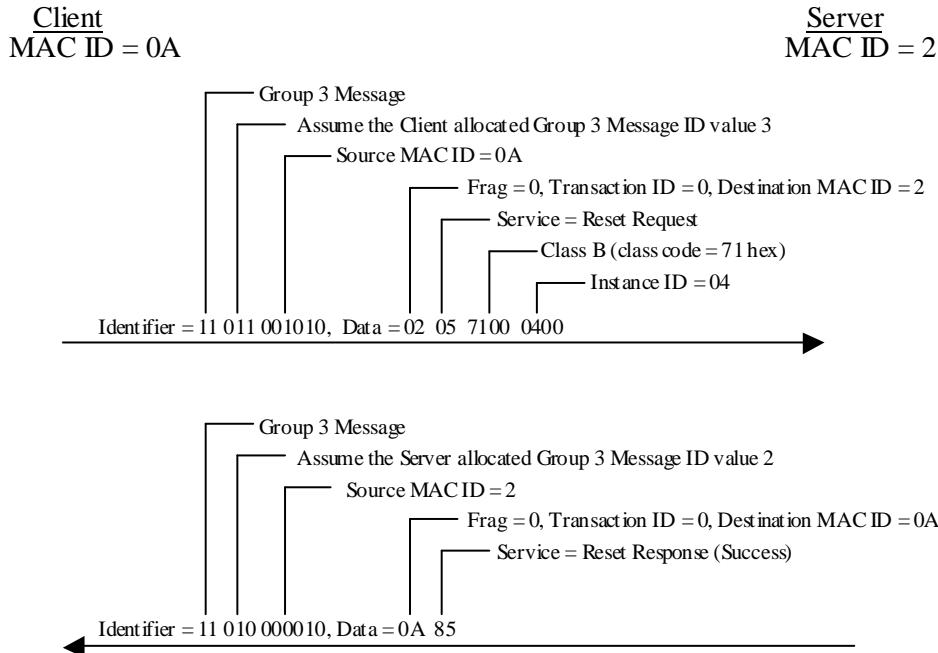
### A-6.2.2 Set\_Attributes\_All

The Client wants to modify all of the attributes from Instance #2 within Class D with one request. Assume the Explicit Messaging Connection was established across Group 3 and the Message Body Format was defined by the Server as DeviceNet (8/8).



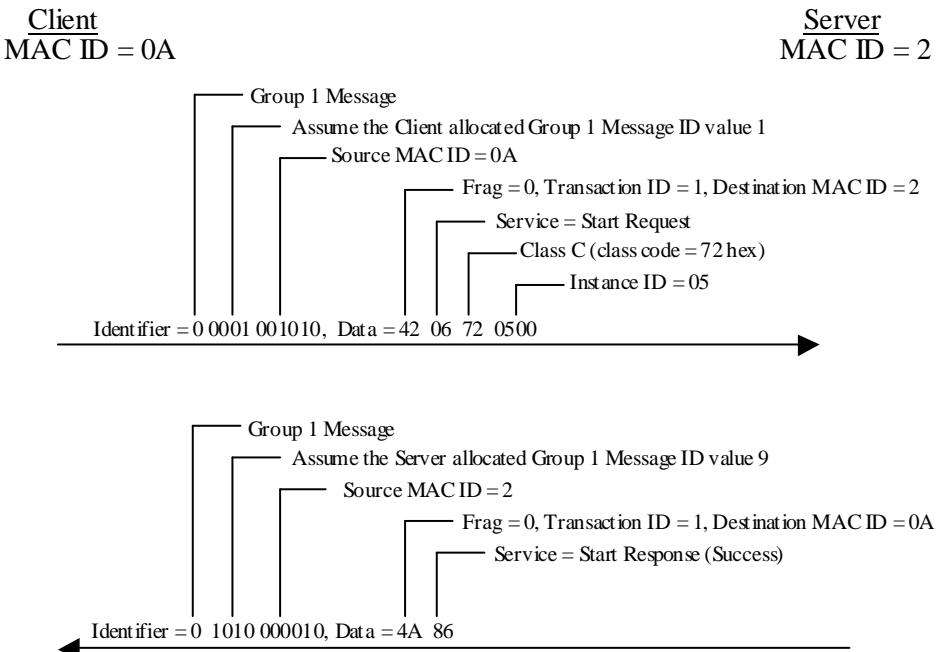
### A-6.2.3 Reset

The Client wants to Reset Instance #4 within Class B. Assume the Explicit Messaging Connection was established across Group 3 and the Message Body Format was defined by the Server as DeviceNet (16/16).



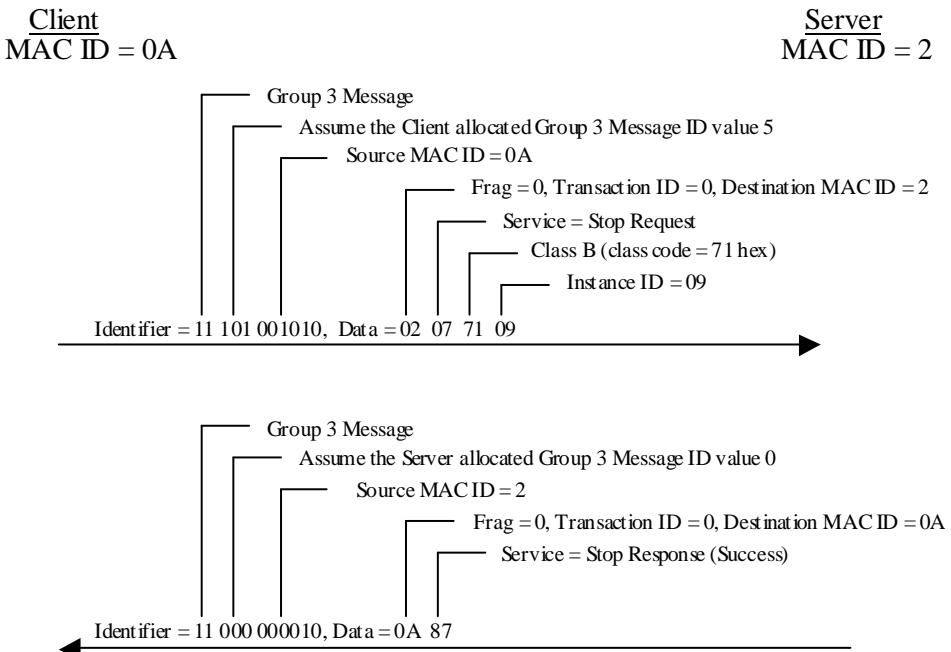
#### A-6.2.4 Start

The Client wants to Start Instance #5 within Class C. Assume the Explicit Messaging Connection was established across Group 1 and the Message Body Format was defined by the Server as DeviceNet (8/16).



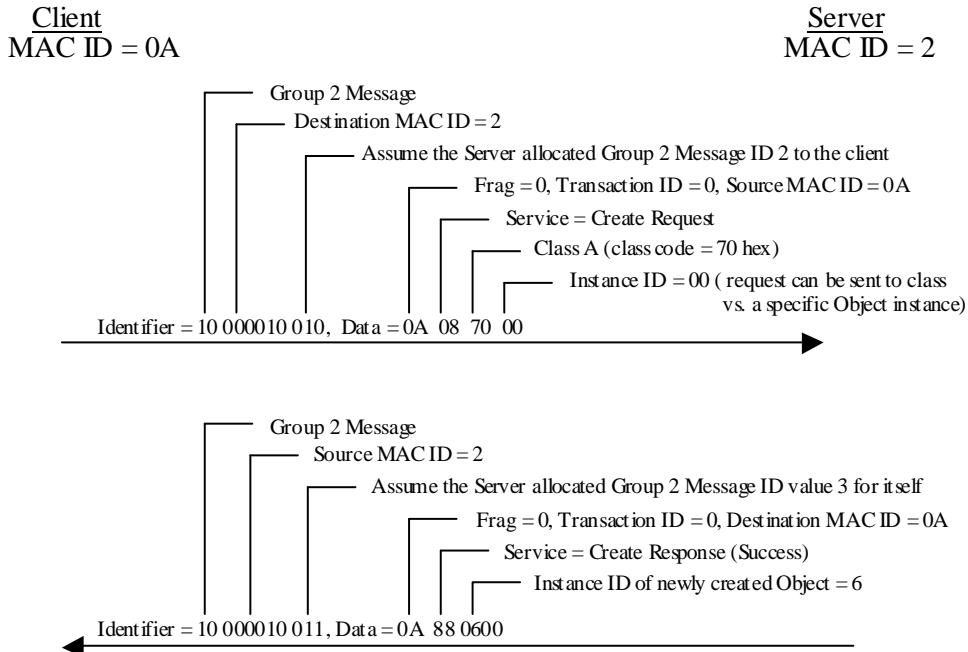
### A-6.2.5 Stop

The Client wants to Stop Instance #9 within Class B. Assume the Explicit Messaging Connection was established across Group 3 and the Message Body Format was defined by the Server as DeviceNet (8/8).



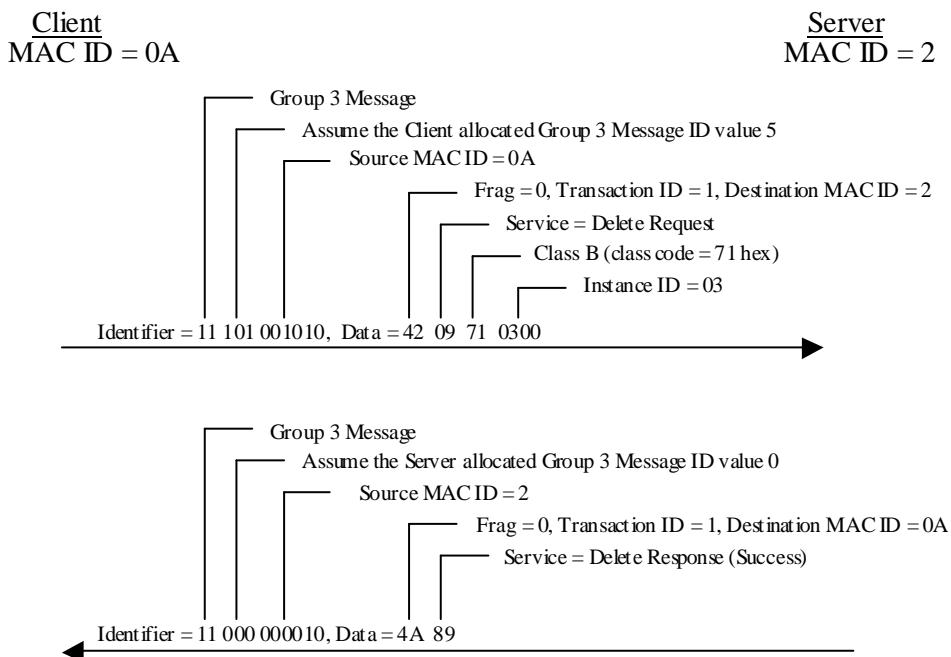
### A-6.2.6 Create

The Client wants to Create an instance within Class A. Assume the Explicit Messaging Connection was established across Group 2 and the Message Body Format was defined by the Server as DeviceNet (8/8)



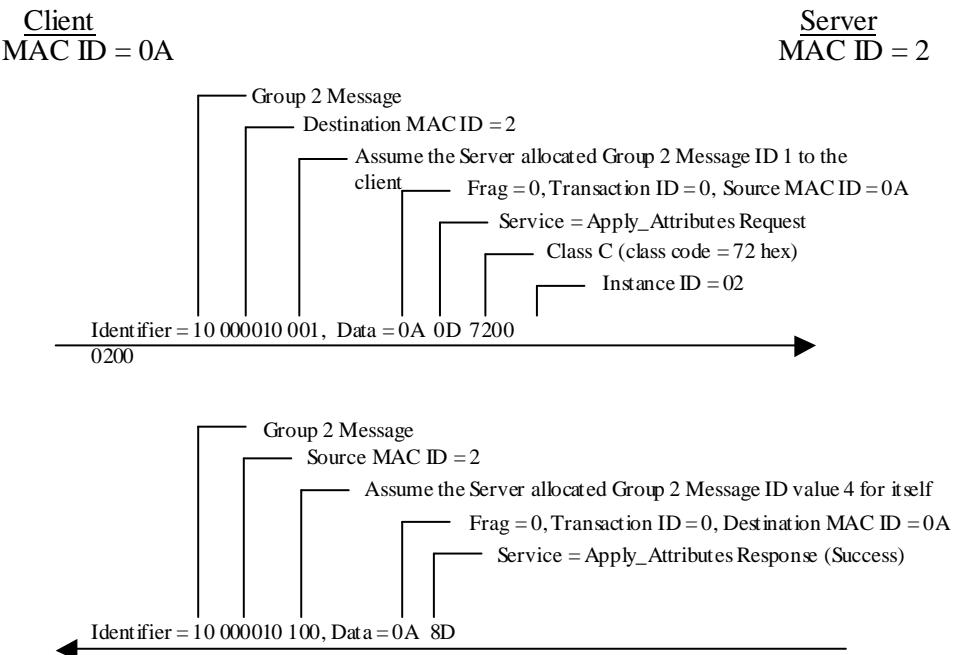
### A-6.2.7 Delete

The Client wants to Delete Instance #3 within Class B. Assume the Explicit Messaging Connection was established across Group 3 and the Message Body Format was defined by the Server as DeviceNet (8/16).



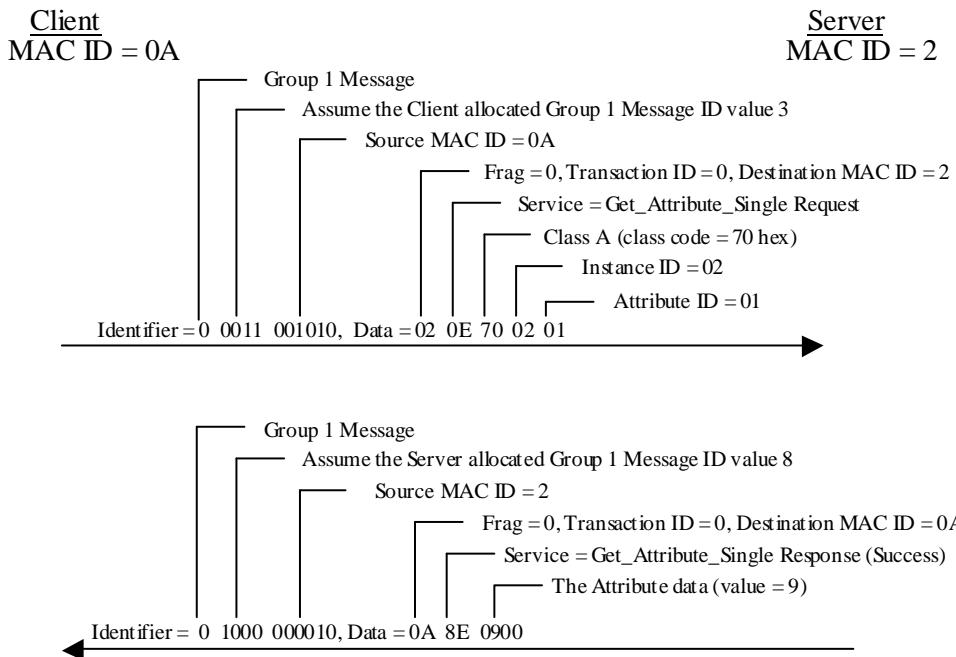
### A-6.2.8 Apply\_Attributes

The Client wants to send an Apply\_Attributes to Instance #2 within Class C. Assume the Explicit Messaging Connection was established across Group 2 and the Message Body Format was defined by the Server as DeviceNet (16/16).



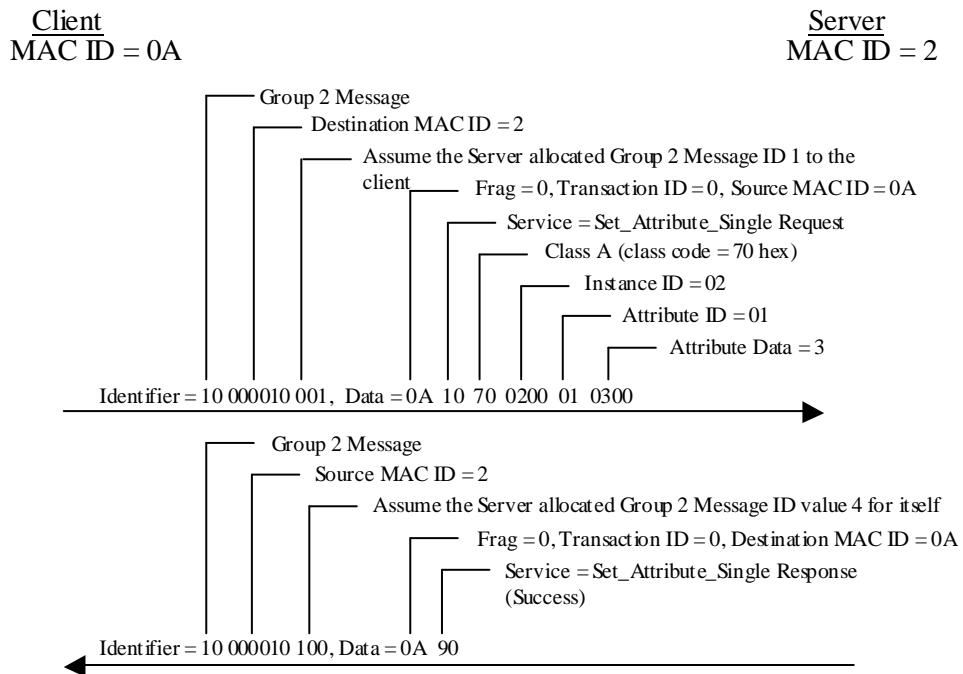
### A-6.2.9 Get\_Attribute\_Single

The Client wants to read Attribute #1 within Instance #2 of Class A. Assume the Explicit Messaging Connection was established across Group 1 and the Message Body Format was defined by the Server as DeviceNet (8/8).



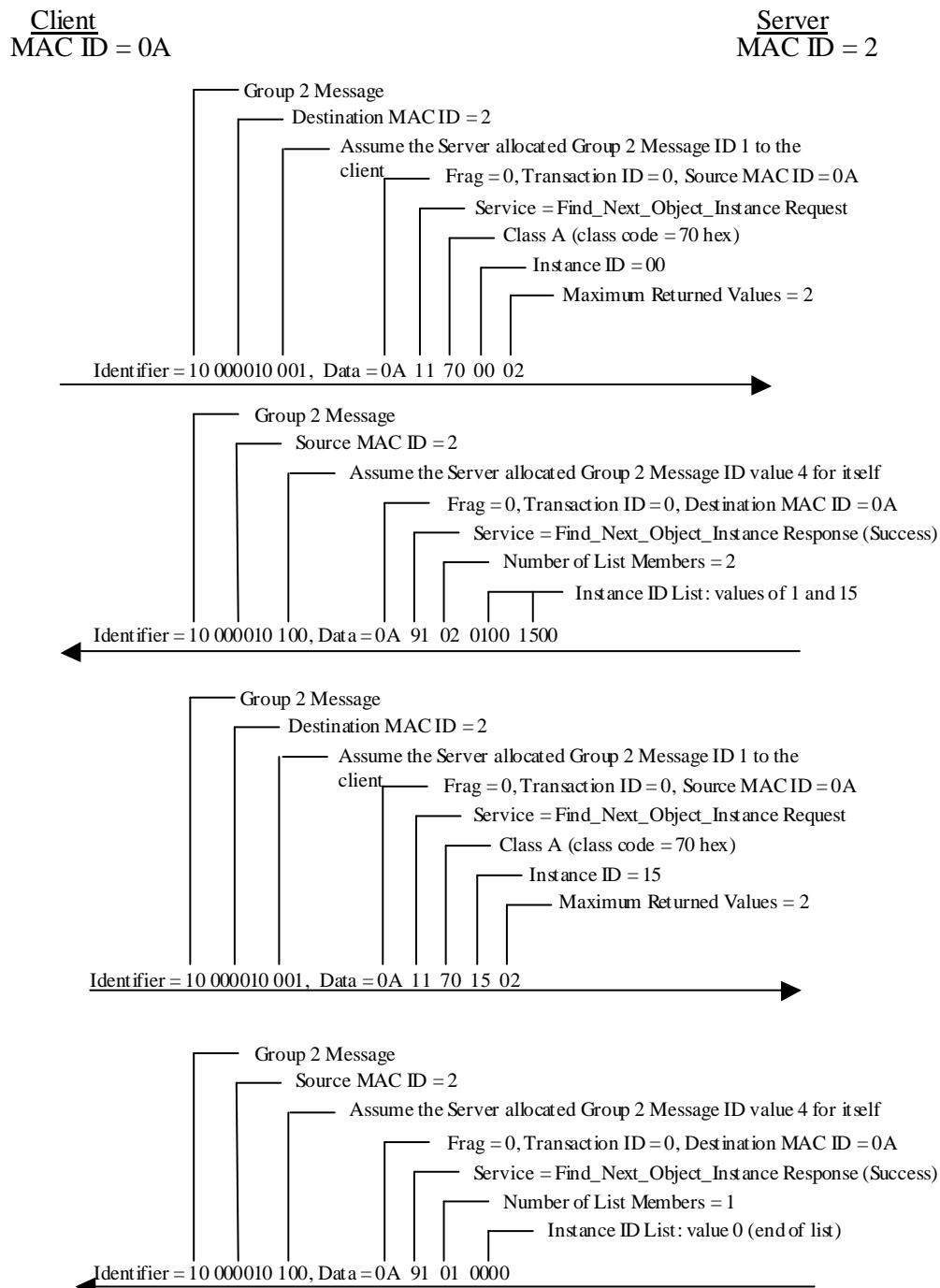
### A-6.2.10 Set\_Attribute\_Single

The Client wants to modify Attribute #1 within Instance #2 of Class A. Assume the Explicit Messaging Connection was established across Group 2 and the Message Body Format was defined by the Server as DeviceNet (8/16).



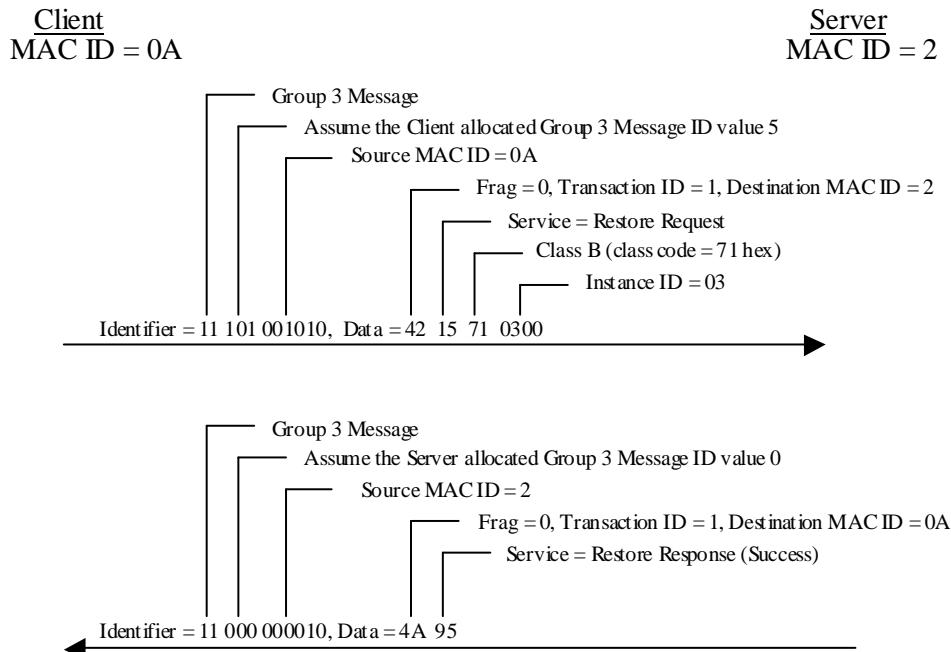
### A-6.2.11 Find\_Next\_Object\_Instance

The Client wants to read the list of Instance IDs associated with existing Objects within Class A. Assume the Explicit Messaging Connection was established across Group 2 and the Message Body Format was defined by the Server as DeviceNet (8/8). Assume also that the Instance IDs 01<sub>hex</sub> and 15<sub>hex</sub> exist.



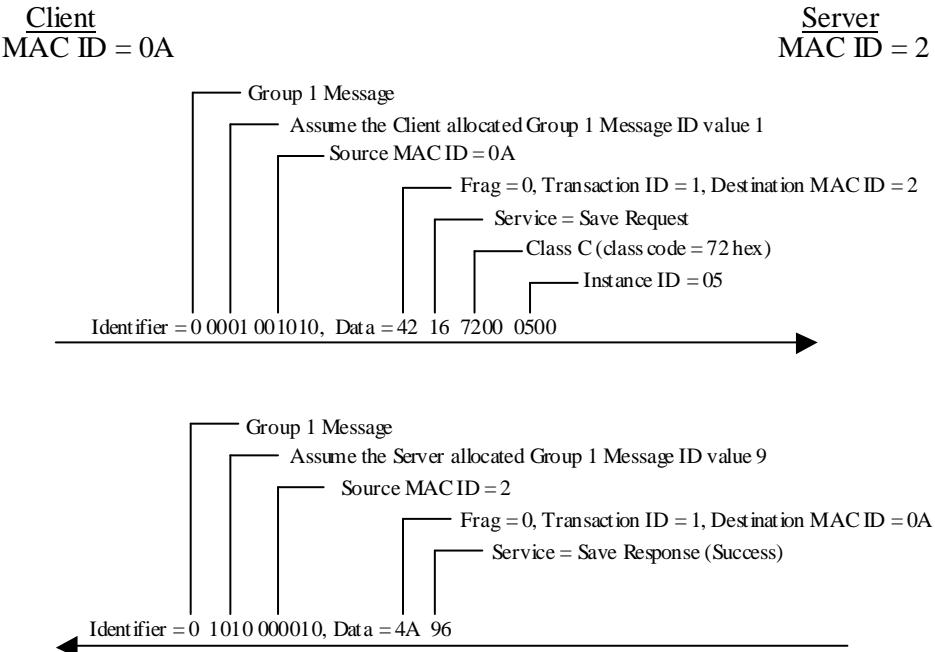
### A-6.2.12 Restore

The Client wants to send a Restore Request to Instance #3 within Class B. Assume the Explicit Messaging Connection was established across Group 3 and the Message Body Format was defined by the Server as DeviceNet (8/8).



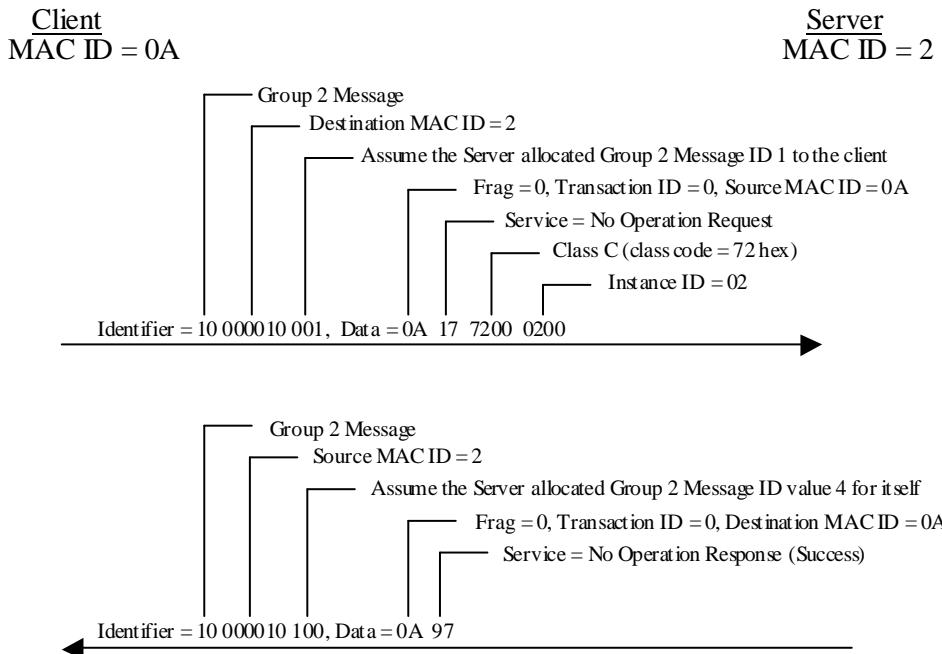
### A-6.2.13 Save

The Client wants to send a Save Request to Instance #5 within Class C. Assume the Explicit Messaging Connection was established across Group 1 and the Message Body Format was defined by the Server as DeviceNet (16/16).



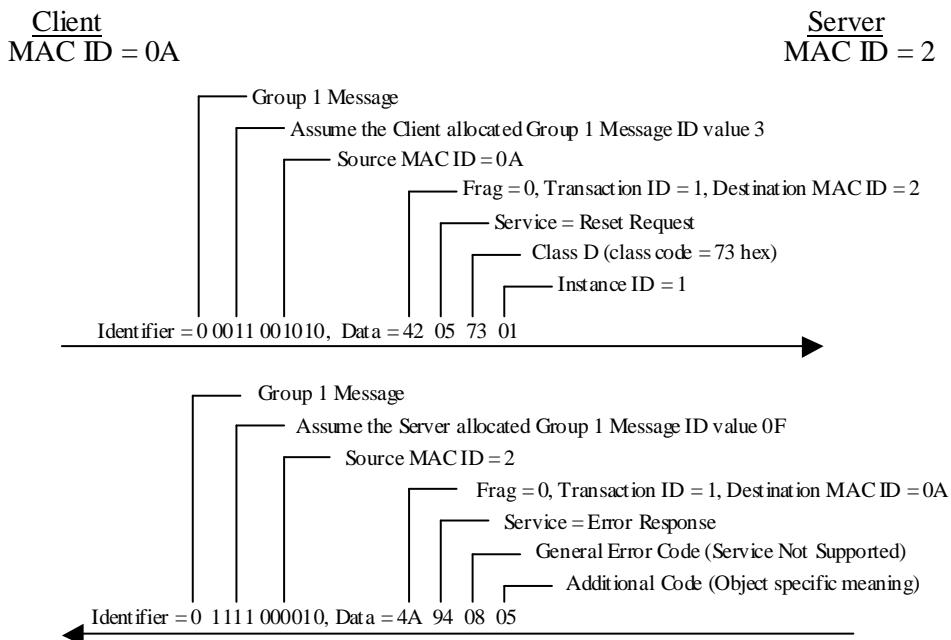
#### A-6.2.14 No Operation

The Client wants to send a No Operation (NOP) to Instance #2 within Class C. Assume the Explicit Messaging Connection was established across Group 2 and the Message Body Format was defined by the Server as DeviceNet (16/16).



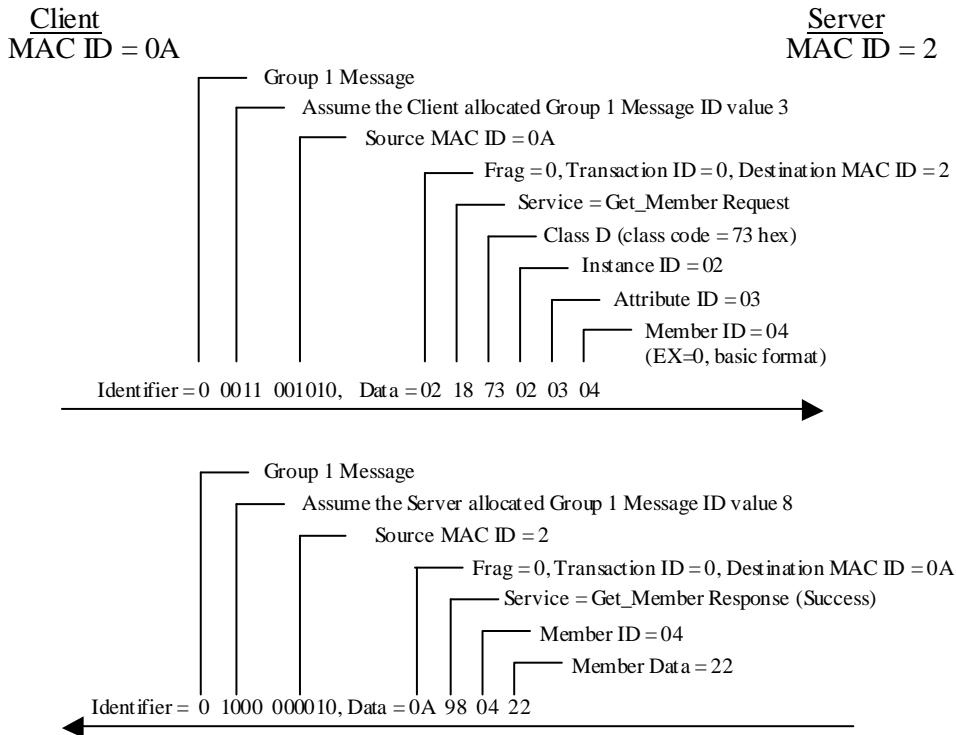
### A-6.2.15 Error Response

The Client wants to Reset Instance #1 within Class D. Assume the Explicit Messaging Connection was established across Group 1 and the Message Body Format was defined by the Server as DeviceNet (8/8). Assume also that Objects within Class D do not support the Reset service.



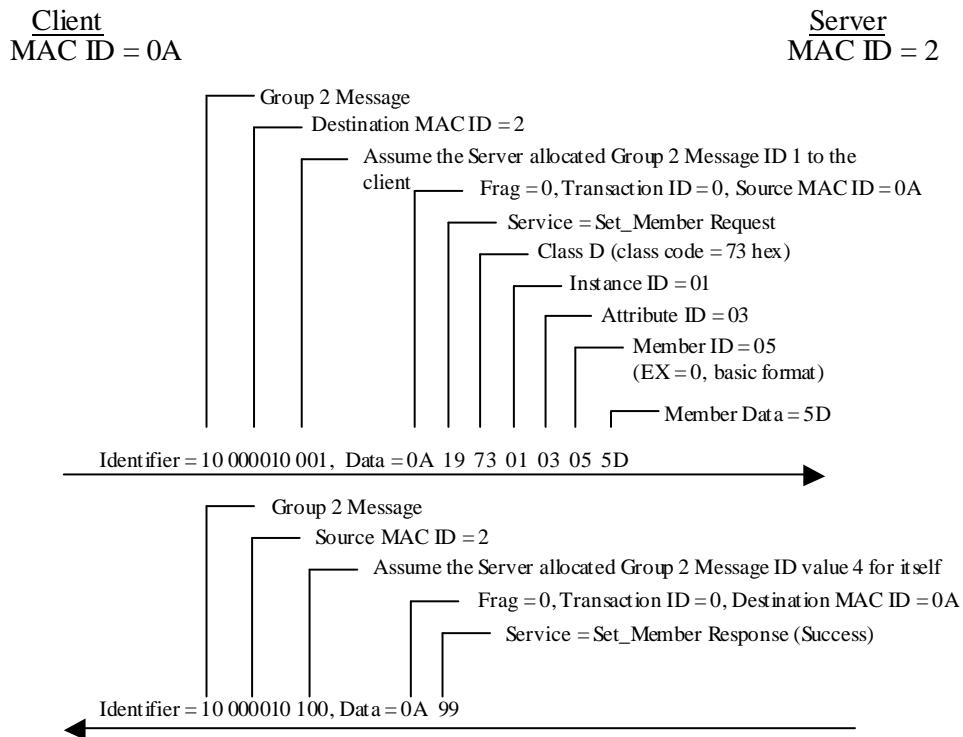
### A-6.2.16 Get\_Member

The Client wants to read Member #4 of Attribute #3 within Instance #2 of Class D. Assume the Explicit Messaging Connection was established across Group 1 and the Message Body Format was defined by the Server as DeviceNet (8/8).



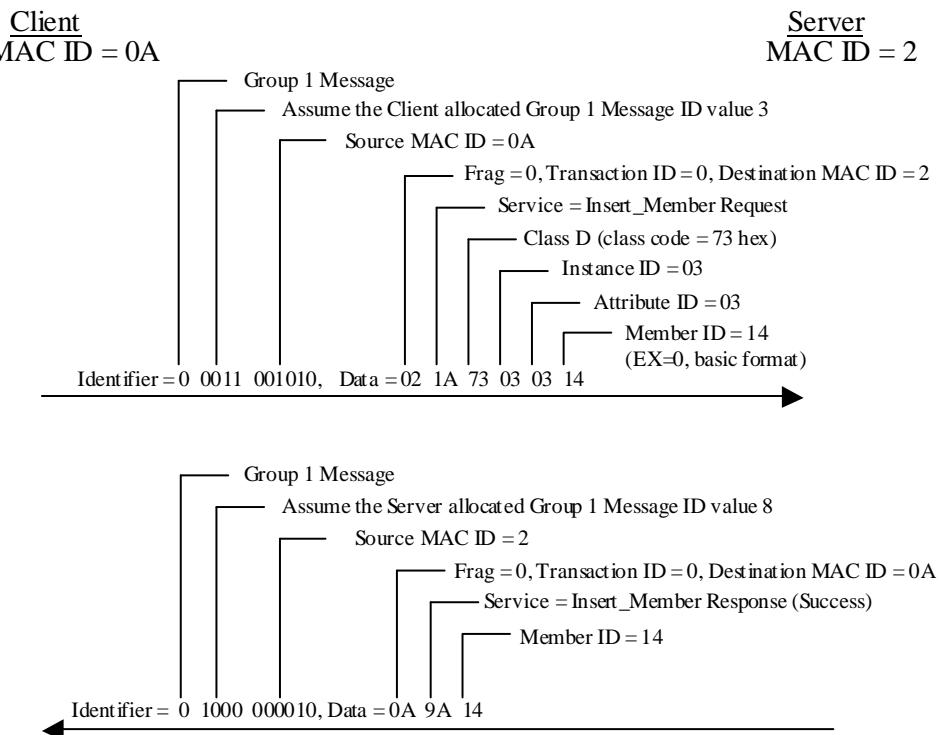
### A-6.2.17 Set\_Member

The Client wants to modify Member #5 of Attribute #3 within Instance #1 of Class D. Assume the Explicit Messaging Connection was established across Group 2 and the Message Body Format was defined by the Server as DeviceNet (8/8).



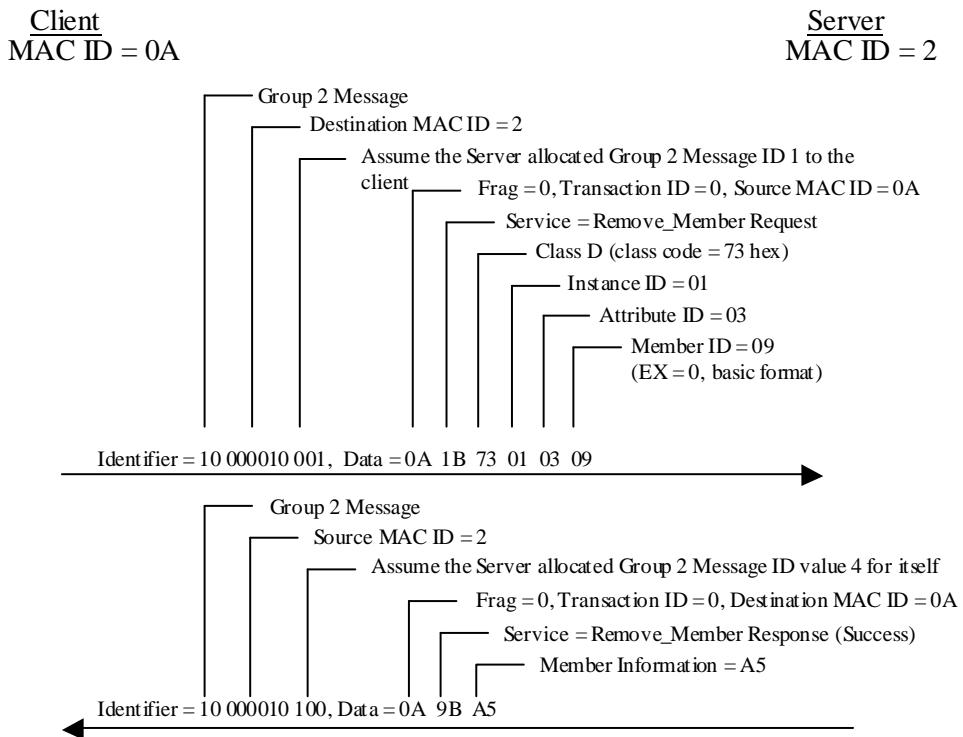
### A-6.2.18 Insert\_Member

The Client wants to Add Member #14 of Attribute #3 within Instance #3 of Class D. Assume the Explicit Messaging Connection was established across Group 1 and the Message Body Format was defined by the Server as DeviceNet (8/8).



### A-6.2.19 Remove\_Member

The Client wants to remove Member #9 of Attribute #3 within Instance #1 of Class D. Assume the Explicit Messaging Connection was established across Group 2 and the Message Body Format was defined by the Server as DeviceNet (8/8).



## **Volume 1: Common Industrial Protocol (CIP<sup>TM</sup>) Specification**

### **Appendix B: Status Codes**

---

## Contents

B-1	General Status Codes .....	3
-----	----------------------------	---

## B-1 General Status Codes

The following table lists the Status Codes that may be present in the General Status Code field of an Error Response message. Note that the Extended Code Field is available for use in further describing any General Status Code. Extended Status Codes are unique to each General Status Code within each object. Each object shall manage the extended status values and value ranges (including vendor specific). All extended status values are reserved unless otherwise indicated within the object definition.

**Table B-1.1 CIP General Status Codes**

General Status Code (in hex)	Status Name	Description of Status
00	Success	Service was successfully performed by the object specified.
01	Connection failure	A connection related service failed along the connection path.
02	Resource unavailable	Resources needed for the object to perform the requested service were unavailable
03	Invalid parameter value	See Status Code 0x20, which is the preferred value to use for this condition.
04	Path segment error	The path segment identifier or the segment syntax was not understood by the processing node. Path processing shall stop when a path segment error is encountered.
05	Path destination unknown	The path is referencing an object class, instance or structure element that is not known or is not contained in the processing node. Path processing shall stop when a path destination unknown error is encountered.
06	Partial transfer	Only part of the expected data was transferred.
07	Connection lost	The messaging connection was lost.
08	Service not supported	The requested service was not implemented or was not defined for this Object Class/Instance.
09	Invalid attribute value	Invalid attribute data detected
0A	Attribute list error	An attribute in the Get_Attribute_List or Set_Attribute_List response has a non-zero status.
0B	Already in requested mode/state	The object is already in the mode/state being requested by the service
0C	Object state conflict	The object cannot perform the requested service in its current mode/state
0D	Object already exists	The requested instance of object to be created already exists.
0E	Attribute not settable	A request to modify a non-modifiable attribute was received.
0F	Privilege violation	A permission/privilege check failed
10	Device state conflict	The device's current mode/state prohibits the execution of the requested service.
11	Reply data too large	The data to be transmitted in the response buffer is larger than the allocated response buffer
12	Fragmentation of a primitive value	The service specified an operation that is going to fragment a primitive data value, i.e. half a REAL data type.
13	Not enough data	The service did not supply enough data to perform the specified operation.
14	Attribute not supported	The attribute specified in the request is not supported
15	Too much data	The service supplied more data than was expected
16	Object does not exist	The object specified does not exist in the device.
17	Service fragmentation sequence not in progress	The fragmentation sequence for this service is not currently active for this data.
18	No stored attribute data	The attribute data of this object was not saved prior to the requested service.
19	Store operation failure	The attribute data of this object was not saved due to a failure during the attempt.

<b>General Status Code (in hex)</b>	<b>Status Name</b>	<b>Description of Status</b>
1A	Routing failure, request packet too large	The service request packet was too large for transmission on a network in the path to the destination. The routing device was forced to abort the service.
1B	Routing failure, response packet too large	The service response packet was too large for transmission on a network in the path from the destination. The routing device was forced to abort the service.
1C	Missing attribute list entry data	The service did not supply an attribute in a list of attributes that was needed by the service to perform the requested behavior.
1D	Invalid attribute value list	The service is returning the list of attributes supplied with status information for those attributes that were invalid.
1E	Embedded service error	An embedded service resulted in an error.
1F	Vendor specific error	A vendor specific error has been encountered. The Additional Code Field of the Error Response defines the particular error encountered. Use of this General Error Code should only be performed when none of the Error Codes presented in this table or within an Object Class definition accurately reflect the error.
20	Invalid parameter	A parameter associated with the request was invalid. This code is used when a parameter does not meet the requirements of this specification and/or the requirements defined in an Application Object Specification.
21	Write-once value or medium already written	An attempt was made to write to a write-once medium (e.g. WORM drive, PROM) that has already been written, or to modify a value that cannot be changed once established.
22	Invalid Reply Received	An invalid reply is received (e.g. reply service code does not match the request service code, or reply message is shorter than the minimum expected reply size). This status code can serve for other causes of invalid replies.
23	<b>Buffer Overflow</b>	<b>The message received is larger than the receiving buffer can handle. The entire message was discarded.</b>
24	<b>Message Format Error</b>	<b>The format of the received message is not supported by the server.</b>
25	Key Failure in path	The Key Segment that was included as the first segment in the path does not match the destination module. The object specific status shall indicate which part of the key check failed.
26	Path Size Invalid	The size of the path which was sent with the Service Request is either not large enough to allow the Request to be routed to an object or too much routing data was included.
27	Unexpected attribute in list	An attempt was made to set an attribute that is not able to be set at this time.
28	Invalid Member ID	The Member ID specified in the request does not exist in the specified Class/Instance/Attribute
29	Member not settable	A request to modify a non-modifiable member was received
2A	Group 2 only server general failure	This error code may only be reported by DeviceNet Group 2 Only servers with 4K or less code space and only in place of Service not supported, Attribute not supported and Attribute not settable.
2B	<b>Unknown Modbus Error</b>	<b>A CIP to Modbus translator received an unknown Modbus Exception Code.</b>
2C - CF		Reserved by CIP for future extensions
D0 - FF	Reserved for Object Class and service errors	This range of error codes is to be used to indicate Object Class specific errors. Use of this range should only be performed when none of the Error Codes presented in this table accurately reflect the error that was encountered.

## **Volume 1: Common Industrial Protocol (CIP<sup>TM</sup>) Specification**

### **Appendix C: Data Management**

## Contents

C-1	Abstract Syntax Encoding Coding for Segment Types.....	3
C-1.1	CIP Segments .....	3
C-1.2	Usage of Segments for Paths .....	3
C-1.3	Path Format .....	4
C-1.4	Segment Type .....	4
C-1.4.1	Port Segment .....	5
C-1.4.2	Logical Segment .....	7
C-1.4.3	Network Segment.....	10
C-1.4.4	Symbolic Segment .....	12
C-1.4.5	Data Segment .....	13
C-1.5	Segment Definition Hierarchy .....	14
C-1.6	Encoded Path Shortcut Rules.....	15
C-2	Data Type Specification.....	15
C-2.1	Data Type Values.....	16
C-2.1.1	Elementary Data Types .....	16
C-2.1.2	Character String Data Types .....	17
C-2.1.3	Structured Bit String Types.....	19
C-2.1.4	Derived Data Types .....	19
C-3	IEC 1131-3 Compliance .....	19
C-3.1	Compliance Statement .....	19
C-3.2	Implementation-Dependent Parameters .....	23
C-3.3	Language Extensions .....	23
C-4	Abstract Syntax Specification.....	24
C-4.1	CIP Data Specification.....	24
C-4.2	Data Type Specification/Dictionaries .....	26
C-5	CIP Application Transfer Syntax: Compact Encoding .....	28
C-5.1	Compact Encoding Constraints.....	29
C-5.2	Encoding Rules/Examples .....	29
C-5.2.1	BOOL Encoding .....	29
C-5.2.2	SignedInteger Encoding.....	29
C-5.2.3	UnsignedInteger Encoding.....	30
C-5.2.4	FixedLengthReal Encoding.....	30
C-5.2.5	Time Encodings .....	31
C-5.2.6	String Encodings.....	31
C-5.2.7	FixedLengthBitString Encoding .....	33
C-5.2.8	Array Encoding.....	33
C-5.2.9	Structure Encoding.....	35
C-5.2.10	Examples of How More Complex Data Formats are Encoded .....	35
C-6	Data Type Reporting.....	36
C-6.1	Elementary Data Type Reporting.....	36
C-6.2	Constructed Data Type Reporting.....	37
C-6.2.1	Structure Type Definition .....	37
C-6.2.2	Array Type Definition.....	39
C-7	CRC Generation Algorithms.....	41

## C-1 Abstract Syntax Encoding Coding for Segment Types

### C-1.1 CIP Segments

This section specifies the encoding for CIP segments. A CIP segment is a stream of encoded items used to reference, describe, and/or configure a specific CIP entity. Segments are usually grouped together in order to provide complete information.

The encoding used for CIP segments has the following relationship with the encoding used to report CIP data types, thus allowing both to be intermixed:

**Table C-1.1 Segment Encoding/Data Reporting Relationships**

Encoding of 1 <sup>st</sup> Byte	Description
00-9F hex	Encoding for Segments
A0-DF hex	Encoding for Data Type Reporting
E0-FF hex	Reserved for future use

A segment contains information indicating what segment type is specified, the format of the segment data, and the actual segment data. Encoding for the following segment types is described in section C-1.4:

- Port segment – used for routing from one subnet to another
- Logical segment - logical reference information (such as class-instance/attribute IDs)
- Network segment - specifies network parameters needed to transmit on a some networks
- Symbolic segment - symbolic name
- Data segment - embedded data (such as configuration data)

Rules for how the segments are ordered and the interpretation of their meaning are described in section C-1.5.

### C-1.2 Usage of Segments for Paths

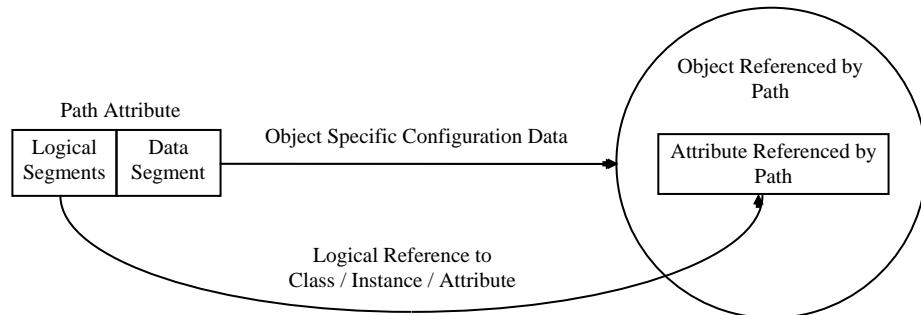
A common use of CIP segments is to specify relationships among different objects. A value used to specify such a relationship is referred to as a path. A path attribute consists of multiple segments, and typically references the class, instance, and attribute of another object. A path has data type EPATH.

Some examples of how paths are used include:

- In Connection and Connection Manager Objects, paths indicate the object(s) to/from which I/O data is moved.
- In Assembly Objects, paths indicate the attributes in other objects which are used to form the assembled I/O data.
- In Parameter Objects, paths indicate the actual attribute in another object which is being described by the Parameter Object.

An example of a path attribute is shown in Figure C-1.1. For an example encoding for such a path, refer to section C-1.4.

**Figure C-1.1 Example Path Attribute**



### C-1.3 Path Format

A path (data type EPATH) can be represented in two different formats:

Padded Path (indicated as data type Padded EPATH)

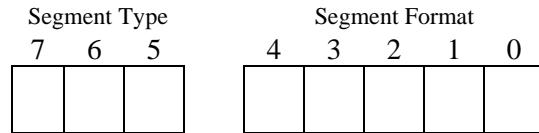
Packed Path (indicated as data type Packed EPATH)

Each segment of a Padded Path shall be 16-bit word aligned. If a pad byte is required to achieve the alignment, the segment shall specify the location of the pad byte. A Packed Path shall not contain any pad bytes. When a component is defined as data type EPATH, it shall indicate the format (Packed or Padded).

### C-1.4 Segment Type

Each segment of an encoding contains a segment type/format byte, which indicates how the segment is to be interpreted. The segment type/format information is contained in the first byte of the segment and, as such, the first byte of any encoding contains a segment type/format byte.

**Figure C-1.2 Segment Type/Format byte Encoding**



The *Segment Type* bits are described below:

Segment Type			
7	6	5	
0	0	0	Port Segment
0	0	1	Logical Segment
0	1	0	Network Segment
0	1	1	Symbolic Segment
1	0	0	Data Segment
1	0	1	Data Type (constructed, see section C-2.1)
1	1	0	Data Type (elementary, see section C-2.1)
1	1	1	Reserved for future use

The meaning of the *Segment Format* bits is based on the specified *Segment Type*.

The Port Segment, Logical Segment, Network Segment, Symbolic Segment, and Data Segment are described in sections C-1.4.1 to C-1.4.5.

#### **C-1.4.1 Port Segment**

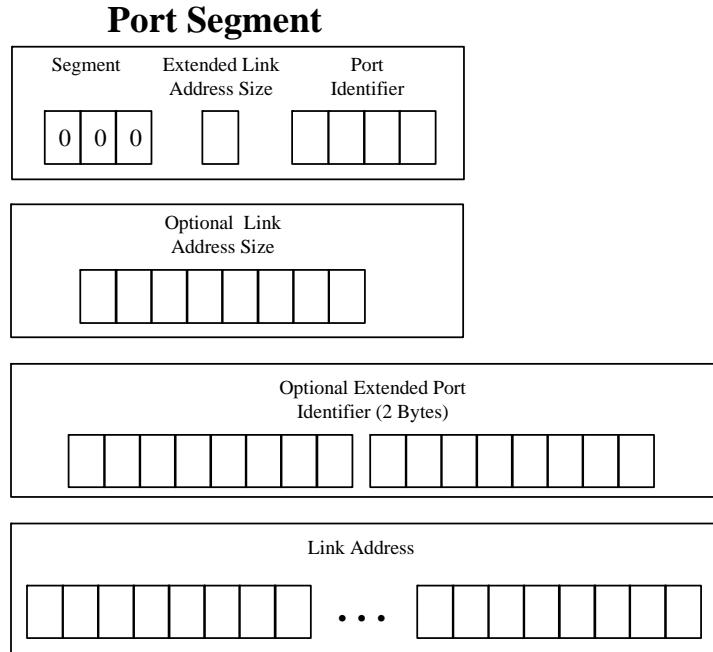
The port segment shall indicate

Communication port through which to leave the node;

Link address of the next device in the routing path.

The Port Segment is formatted as defined in the figure below. The bit representation is from high bit to low bit, left to right. The byte representation is from low byte to high byte, top to bottom and left to right.

**Figure C-1.3 Port Segment Encoding**



Bit 4, the **Extended Link Address Size** bit, shall be set to 0 if the link address is one byte. Bit 4 shall be set to 1 if the link address is larger than one byte. If the link address is larger than one byte, its size in bytes shall be in the second byte of the Port Segment.

Bits 0 – 3, the **Port Identifier**, shall indicate through which port to leave the node. The Port Identifier shall specify a *Port Number* or an escape to an extended port identifier when the module can support more than 14 ports. The list below defines rules pertinent to Port Identification/Port Number and Link Address values:

Port Number 0 shall be reserved.

Port Number 1 shall only be used to represent a back-plane port.

If the Port Identifier is 15, then a 16-bit field, called the Extended Port Identifier, shall be the next part of the Port Segment (following the optional Link Address Size if present); otherwise, the value of the Port Identifier shall be the Port Number.

The Port Number shall be followed by a **Link Address** whose format depends on the type of network to which the Port Identifier refers. If the Link Address is greater than one byte it shall be padded so that the length of the entire port segment is an even number of bytes. The pad byte shall be set to zero and shall not be included in the Link Address Size. With respect to the specification of a DeviceNet Port, the Link Address shall indicate the target DeviceNet MAC ID and be specified in a single byte whose value is 0 – 63 (the valid range of DeviceNet MAC IDs).

The Extended Port Identifier format of the Port Segment shall only be used when there are more than 14 network ports possible on the connecting device. The Port Segment shall always be represented in the smallest Port Segment format possible with respect to the optional fields.

**Examples** of Port Segments are presented below.

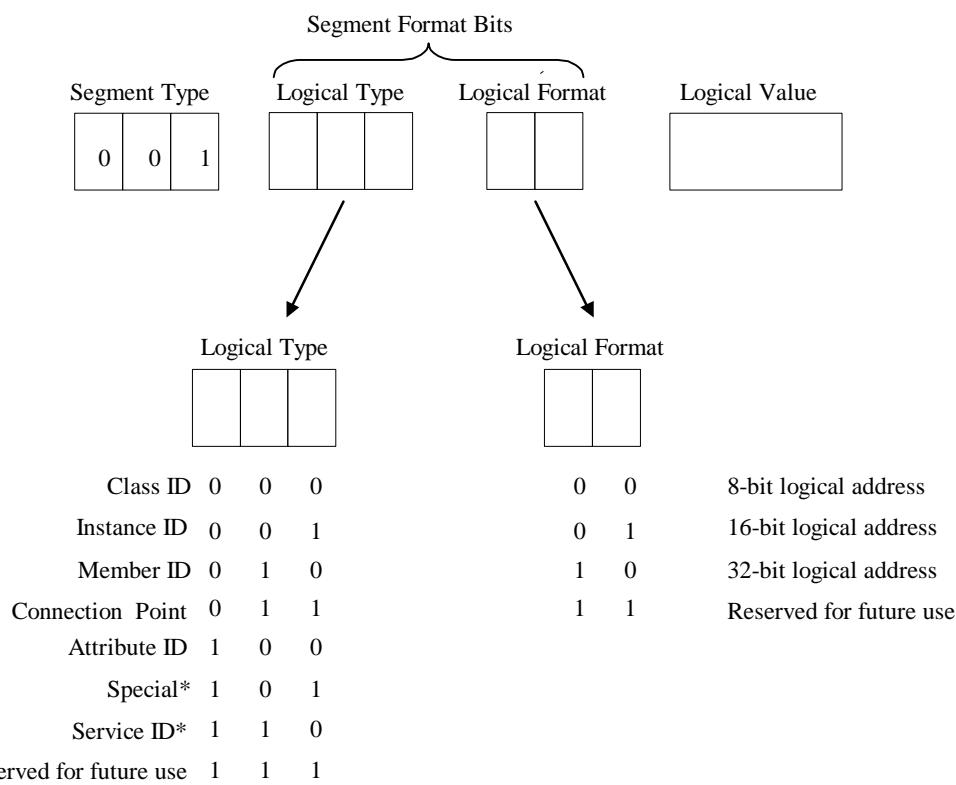
**Table C-1.2 Port Segment Examples**

EPAUTH contents	Notes
[02][06]	Segment Type = Port Segment. Port Number = 2, Link Address = 6
[0F][12][00][01]	Segment Type = Port Segment. Port Identifier is 15 indicating the Port Number is specified in the next 16 bit field [12][00] (18 decimal). Link Address = 1.
[15][0F][31][33][30][2E] [31][35][31][2E][31][33] [37][2E][31][30][35][00]	Segment Type = Port Segment. Multi-Byte address for TCP Port 5, Link Address 130.151.137.105 (IP Address). The address is defined as a character array, length of 15 bytes. The last byte in the segment is a pad byte.

#### C-1.4.2 Logical Segment

The logical segment selects a particular object address within a device (for example, Object Class, Object Instance, and Object Attribute). When the logical segment is included within a Packed Path, the Logical Value shall be appended to the segment type byte with no pad in between. When the logical segment is included within a Padded Path, the 16-bit and 32-bit logical formats shall have a pad inserted between the segment type byte and the Logical Value (the 8-bit format is identical to the Packed Path). The pad byte shall be set to zero.

**Figure C-1.4 Logical Segment Encoding**



\*The Special and Service ID Logical Types do not use the logical addressing definition for the Logical Format.

The 8-bit and 16-bit logical address formats are allowed for use with all Logical Types.

The 32-bit logical address format is **only allowed for the logical Instance ID and Connection Point types. It is not allowed for any other Logical Type** (reserved for future use).

The *Connection Point* Logical Type provides additional addressing capabilities beyond the standard Class ID/Instance ID/Attribute ID/Member ID Object Address. *Object Classes shall define when and how this addressing component is utilized.*

The *Service ID* Logical Type has the following definition for the Logical Format:

- 0 0 8-Bit Service ID Segment (0x38)
- 0 1 Reserved for future use (0x39)
- 1 0 Reserved for future use (0x3A)
- 1 1 Reserved for future use (0x3B)

The *Special* Logical Type has the following definition for the Logical Format:

- 0 0 Electronic Key Segment (0x34)
- 0 1 Reserved for future use (0x35)
- 1 0 Reserved for future use (0x36)
- 1 1 Reserved for future use (0x37)

The *Electronic Key* segment shall be used to verify/identify a device. Possible uses include verification during connection establishment and identification within an EDS file. This segment has the format as shown in the table below.

**Table C-1.3 Electronic Key Segment Format**

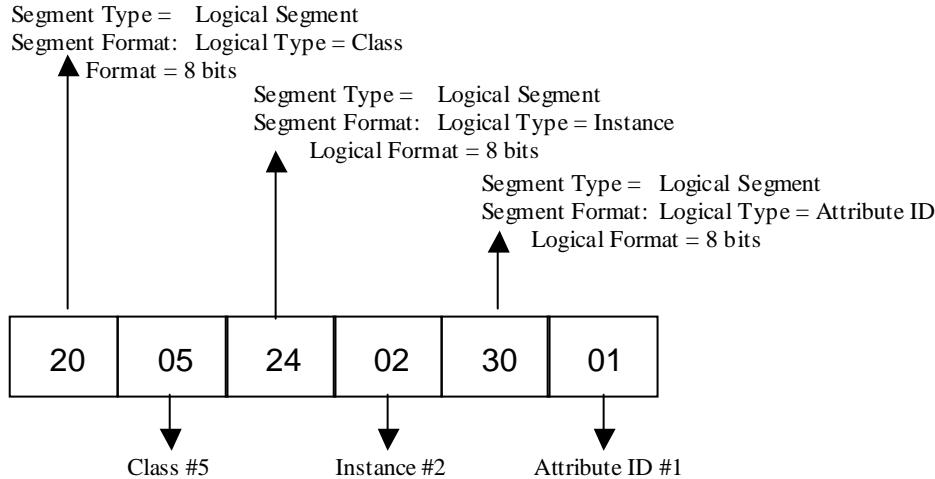
Field Name	Data Type	Semantics
Segment Type	USINT	A value of 0x34 indicates a Logical Electronic Key segment
Key Format	USINT	0 – 3 = Reserved 4 = see Key Format Table 5 – 255 = Reserved
Key Data	Array of octet	Depends on key format used

**Table C-1.4 Key Format Table**

Format Value	Field Name	Data Type	Semantics
4	Vendor ID	UINT	Vendor ID
	Device Type	UINT	Device Type
	Product Code	UINT	Product Code
	Major Revision / Compatibility	BYTE	Bits 0 – 6 = Major Revision Bit 7 = Compatibility (If clear then any non-zero Vendor ID, Device Type, Product Code, Major Revision, and Minor Revision shall match. If set, then any key may be accepted which a device can emulate.)
	Minor Revision	USINT	Minor Revision

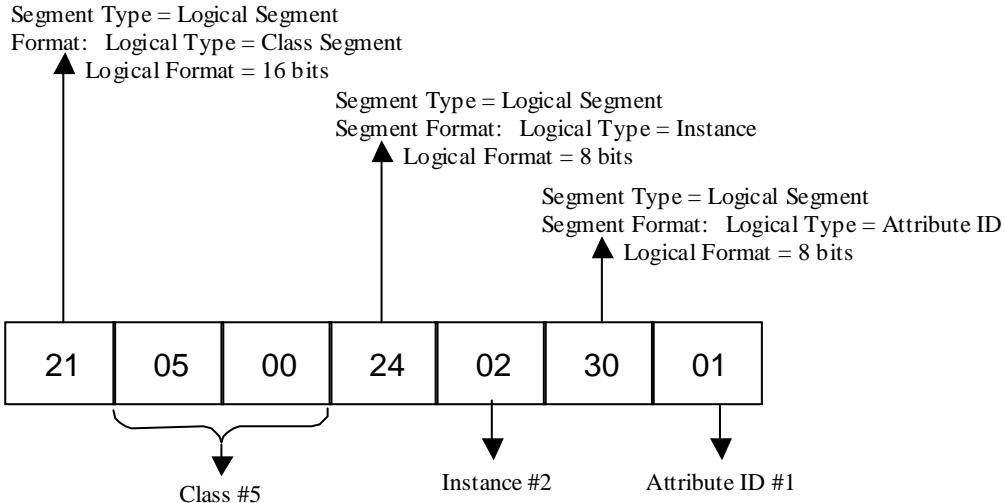
An encoding that specifies Class #5, Instance #2, and Attribute ID #1 is illustrated below (Note that the packed and padded representations are the same):

**Figure C-1.5 Packed EPATH with 8 bit Class**



The same example is presented below, except that the Class information is specified in a 16 bit field. Figure C-1.6 shows a packed representation and Figure C-1.7 shows padded.

**Figure C-1.6 Packed EPATH with 16 bit Class**



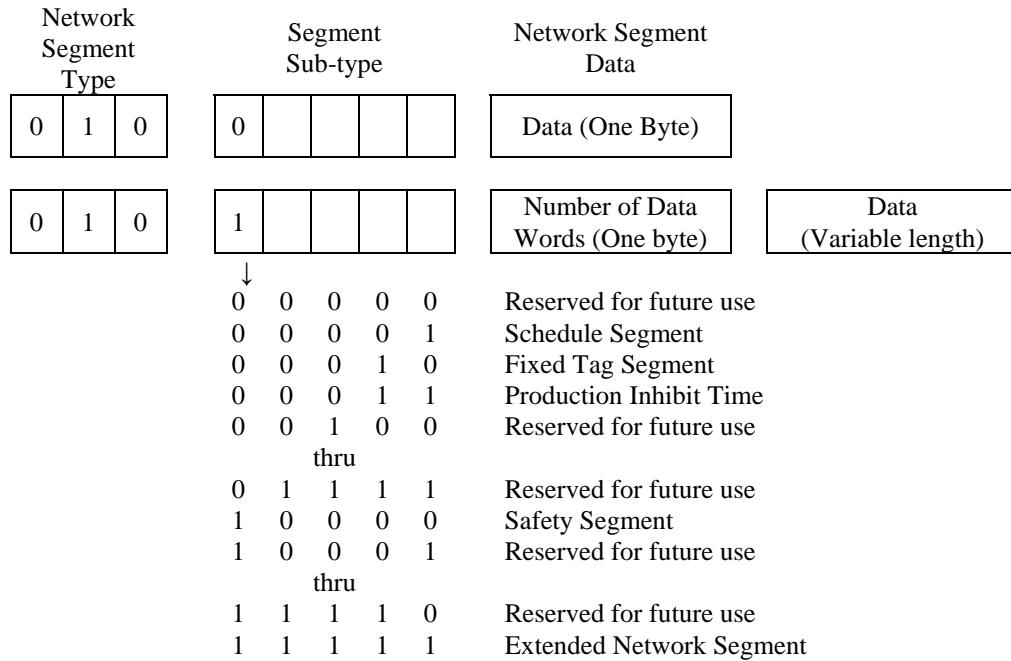
**Figure C-1.7 Padded EPATH with 16 bit Class**

21	00	05	00	24	02	30	01
▲ Pad inserted here for padded representation							

### C-1.4.3 Network Segment

The network segment shall be used to specify network parameters that may be required by a node to transmit a message across a network. The network segment shall immediately precede the port segment of the device to which it applies. In other words, the network segment shall be the first item in the path that the device receives.

**Figure C-1.8 Network Segment Encoding**



#### C-1.4.3.1 Schedule Network Segment

Appendix C-2.1 of Volume 4 (ControlNet Adaptation of CIP) defines the Schedule network segment

#### C-1.4.3.2 Fixed Tag Network Segment

Appendix C-2.2 of Volume 4 (ControlNet Adaptation of CIP) defines the Fixed Tag network segment.

#### C-1.4.3.3 Production Inhibit Time Network Segment

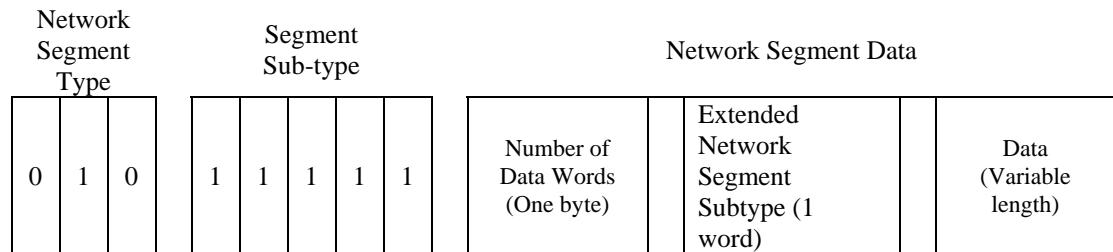
The production inhibit time network segment shall specify the minimum time, in milliseconds, between successive transmissions of connected data for the specified connection. For example, if a production inhibit time of 10 milliseconds is specified, new data shall be sent no sooner than 10 milliseconds after the previous data. The network segment data field shall be a single USINT (8 bits) allowing for a range of 1 – 255 milliseconds. A value of zero shall indicate no production inhibit time. When this segment does not exist during establishment of a connection a default value of one-fourth the RPI shall be used.

The RPI shall be larger than the production inhibit time. If the RPI is smaller than the production inhibit time, the Forward\_Open\_reply shall be returned with error (status 0x01, extended status 0x111).

#### **C-1.4.3.4 Extended Network Segment**

The Extended Network Segment allows for the definition of additional Network Segment Subtypes. The first word of data is the Extended Network Segment Subtype.

**Figure C-1.9 Extended Network Segment Format**



Each extended subtype defines the data that follows. The extended subtypes are enumerated in the table below.

**Table C-1.5 Extended Subtype Value Definition**

Extended Subtype Value	Extended Subtype Definition
0 – 65535	Reserved for future use

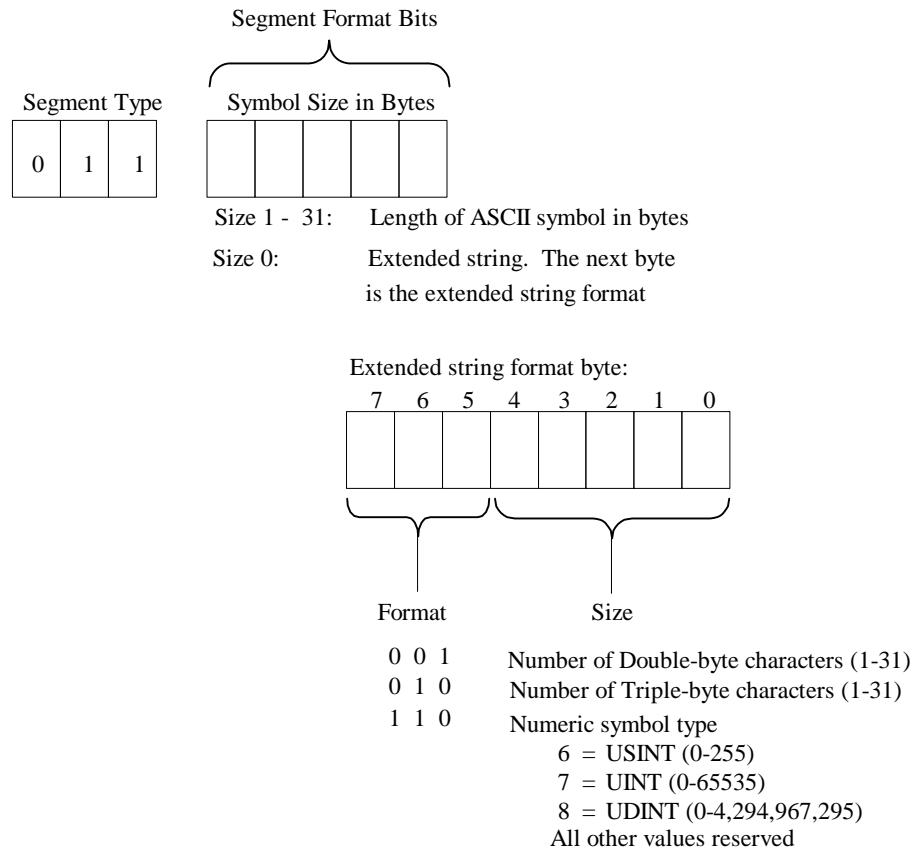
#### **C-1.4.3.5 Safety Network Segment**

The *Safety* network segment is defined in the CIP Safety Specification (Volume 5, Appendix C)

#### C-1.4.4 Symbolic Segment

The symbolic segment contains an International String symbol which must be interpreted by the device.

**Figure C-1.10 Symbolic Segment Encoding**



Character Symbols can contain a maximum of 31 characters.

Numeric Symbols can be 8, 16, or 32 bits.

For example (values are hexadecimal):

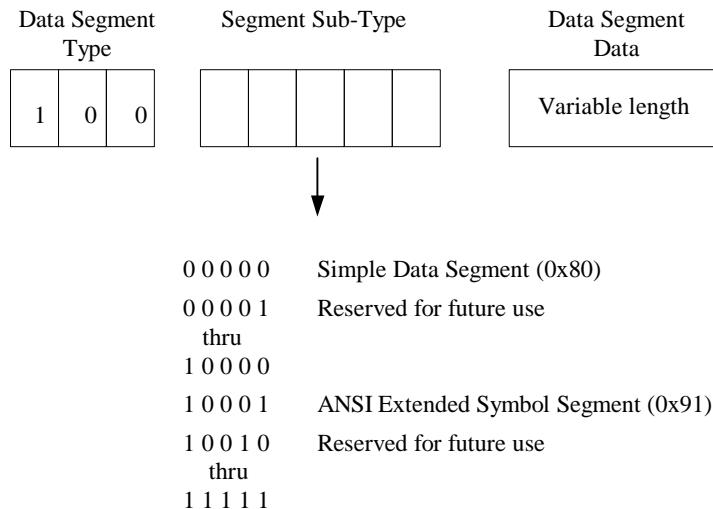
**Table C-1.6 Symbolic Segment Examples**

Symbolic Segment	Notes
[65][LS101]	ASCII Symbol. This could refer to a bit in a data structure
[67][Line_23]	ASCII Symbol. This could refer to a controller in a slot
[68][Wire_off]	ASCII Symbol. This could refer to a bit in a diagnostic structure
[60][22][1234][2345]	Japanese symbol
[60][C7][1234]	16 bit Numeric Symbol
[60][C8][12345678]	32 bit Numeric Symbol

### C-1.4.5 Data Segment

The data segment provides a mechanism for delivering data to an application. This may occur during connection establishment, or at any other time as defined by the application.

**Figure C-1.11 Data Segment Encoding**



#### C-1.4.5.1 Simple Data Segment

The Simple data segment contains data values such as parameters for the target application. The size of the data segment is specified in number of 16 bit words and depends on the application. This byte value immediately follows the segment type. The data values follow the length byte. The data segment can be any number of 16 bit words up to 255. An encoding can contain more than one data segment if required.

The following table provides examples of data segments:

**Table C-1.7 Data Segment Encoding**

Encoding	Notes
[80][07] [0100] [0200] [0300] [0400] [0500] [0600] [0700]	Seven words of data in a data segment
[21] [0500] [24] [09] [80][04] [0100] [0200] [0300] [0400]	Logical segments for Class 5, Instance 9, then four words of data in a data segment

The data can be configuration information for the object, additional parameters necessary for the object, or any other set of object specific information.

#### C-1.4.5.2 ANSI Extended Symbol Segment

The segment type of ANSI extended symbol segment shall be 0x91. The byte following the segment type (second byte) shall represent the number of characters (8-bit) in the symbol (symbol size). The variable length symbol shall follow the symbol size and shall end with a pad byte of zero if the size is of odd length. The symbol size shall not count the pad byte if it is included.

The following table provides examples of extended symbol segments:

**Table C-1.8 ANSI Extended Symbol Segment Examples**

Encoding	Notes
[91][06] [73] [74] [61] [72] [74] [31]	Six bytes of data in the symbol ‘start1’
[91] [07] [73] [74] [61][72] [74] [65] [72] [00]	Seven bytes of data in the symbol ‘starter’ (plus a null pad)

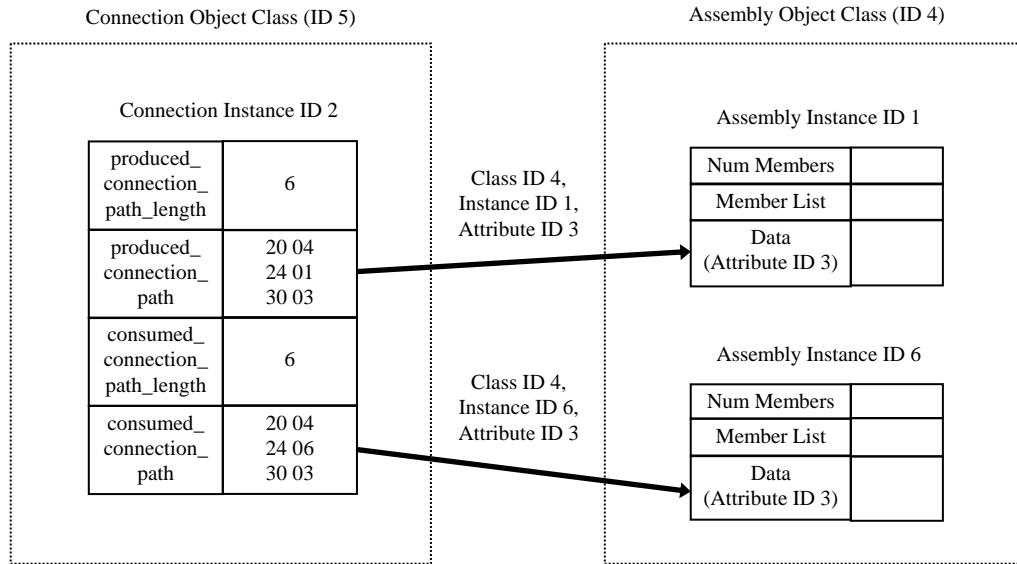
### C-1.5 Segment Definition Hierarchy

The definition of any rules related to the use of segments is defined within the Object Class and/or Device Profile. The following are examples of valid recommended hierarchies. The depth within the hierarchy need only proceed to the degree required by its application.

- Class ID,
- Class ID, Instance ID
- Class ID, Instance ID, Attribute ID
- Class ID, Instance ID, Attribute ID, Member ID
  
- Class ID, Connection Point
- Class ID, Connection Point, Member ID
- Class ID, Instance ID, Connection Point
- Class ID, Instance ID, Connection Point, Member ID
- Symbolic ID
- Symbolic ID, Member ID
- **Symbolic ID, Connection Point**

The following example demonstrates encoding for two path attributes: the **produced\_connection\_path** and **consumed\_connection\_path** attributes of the Connection Object.

**Figure C-1.12 Example Encoding for Connection Object Paths**



## C-1.6 Encoded Path Shortcut Rules

When multiple encoded paths are concatenated the delineation between paths is where a segment at a higher level in the hierarchy is encountered. Multiple encoded paths may be compacted when each path shares the same values at the higher levels in the hierarchy. When a segment is encountered which is at the same or higher level but not at the top level in the hierarchy, the preceding higher levels are used for that next encoded path. The examples below show multiple encoded paths in the full and compacted representations.

Full: Class A, Instance A, Attribute A, Class A, Instance A, Attribute B

Compact: Class A, Instance A, Attribute A, Attribute B

Full: Class A, Instance A, Attribute A, Class A, Instance B, Attribute A

Compact: Class A, Instance A, Attribute A, Instance B, Attribute A

## C-2 Data Type Specification

This section describes the data type specification syntaxes, data type value ranges, and operations that can be performed on the defined data types. The specification of a data type comprises:

- the range of *values* that variables of the type may take on, and
- the *operations* performed on these variables

This CIP standard specifies *elementary* and *derived* data types corresponding to the notation of IEC 1131-3. In addition, since function blocks as defined in IEC 1131-3 have both an associated data structure and a set of defined standard operations, these elements are also specified as data types in this CIP standard.

## C-2.1 Data Type Values

Data is made up of *elementary* (primitive) data types. These elementary data types are used to construct *derived* (constructed) data types.

### C-2.1.1 Elementary Data Types

The elementary data types and the values (range) of variables of each type are given in Table C-2.1 and its associated notes. This table specifies the values of certain parameters which are identified as “implementation–dependent” in Table 10.1 of subclause 2.3.1 in IEC 1131-3. The implementation–dependent parameters defined in this CIP Common specification can be found in Table C-3.4.

**Table C-2.1 Elementary Data Types**

Keyword	Description	Range	
		Minimum	Maximum
BOOL	Boolean		NOTE 1
SINT	Short Integer	-128	127
INT	Integer	-32768	32767
DINT	Double Integer	$-2^{31}$	$2^{31}-1$
LINT	Long Integer	$-2^{63}$	$2^{63}-1$
USINT	Unsigned Short Integer	0	255
UINT	Unsigned Integer	0	65535
UDINT	Unsigned Double Integer	0	$2^{32}-1$
ULINT	Unsigned Long Integer	0	$2^{64}-1$
REAL	Floating point		NOTE 2
LREAL	Long float		NOTE 3
ITIME	Duration (short)		NOTE 12
TIME	Duration		NOTE 4
FTIME	Duration (high resolution)		NOTE 5,6
LTIME	Duration (long)		NOTE 6,7
DATE	Date only		NOTE 8
TIME_OF_DAY or TOD	Time of day		NOTE 9
DATE_AND_TIME or DT	Date and time of day		NOTE 10
STRING	character string (1 byte per character)		
STRING2	character string (2 bytes per character)		NOTE 6
STRINGN	character string (N-bytes per character)		NOTE 6
SHORT_STRING	character string (1 byte per character, 1 byte length indicator)		NOTE 6
STRINGI	International character string		NOTE 6
BYTE	bit string - 8 bits		NOTE 11
WORD	bit string - 16-bits		NOTE 11
DWORD	bit string - 32-bits		NOTE 11
LWORD	bit string - 64-bits		NOTE 11
EPAUTH	CIP path segments		NOTE 13
ENGUNIT	Engineering Units		NOTE 14

<sup>1</sup> The possible values of variables of type BOOL 0 and 1, corresponding to the keywords FALSE and TRUE, respectively.

Keyword	Description	Range	
		Minimum	Maximum
2	The range of values for variables of type REAL are defined in IEEE 754 for the basic single floating-point format.		
3	The range of values for variables of type LREAL are defined in IEEE 754 for the basic double floating-point format.		
4	The range of values for variables of type TIME is the same as for variables of type DINT, representing the time duration in milliseconds, i.e., a range of T#-24d20h31m23.648s to T#24d20h31m23.647s.		
5	The range of values for variables of type FTIME is the same as for variables of type DINT, representing the time duration in microseconds, i.e., a range of T#-35m47.483648s to T#35m47.483547s.		
6	This is a CIP standard extension to IEC 1131-3.		
7	The range of values for variables of type LTIME is the same as for variables of type LINT, representing the time duration in microseconds, i.e., a range of T#-106751991d4h0m54.775808s to T#106751991d4h0m54.775807s.		
8	The range of values for variables of type DATE is from D#1972-01-01, the start of the Coordinated Universal Time (UTC) era, to D#2151-06-06 (a total range of 65536 days).		
9	The range of values for variables of type TIME_OF_DAY is from TOD#00:00:00.000 to TOD#23:59:59.999 to a resolution of 1 millisecond.		
10	The range of values for variables of type DATE_AND_TIME is from DT#1972-01-01-00:00:00.000 to DT#2151-06-06-23:59:59.999.		
11	Values of bit string data types is in the range 2#b <sub>N-1</sub> b <sub>N-2</sub> ...b <sub>2</sub> b <sub>1</sub> b <sub>0</sub> , where N is the number of bits in the bit string, b <sub>N-1</sub> is the “most significant bit”, and b <sub>0</sub> is the “least significant bit”. The value of the j-th bit b <sub>j</sub> is represented as 0 or 1, corresponding to the Boolean value FALSE or TRUE, respectively.		
12	The range of values for variables of type ITIME is the same as for variables of type INT, representing the time duration in milliseconds, i.e., a range of T#-32s768ms to T#32s767ms.		
13	For complete information on the EPATH data type, refer to Appendix C: Abstract Syntax Encoding for Segment Types.		
14	The range of values for variables of type ENGUNIT is the same as for variables uint, representing the values enumerated in Appendix D of the CIP specification.		

## C-2.1.2

### Character String Data Types

The declaration of a variable of type STRING, STRING2, or STRINGN is equivalent to declaring a structured data type for the variable which allocates a UINT variable containing the current size of the string in characters and an array of declared character size elements.

STRING declares 8-bit octet elements. STRING2 declares 16-bit octet elements. STRINGN declares N\*8-bit octets. STRINGN includes a UINT variable which declares the character size. Character strings of data type STRING shall be based on ISO-8859-1. Character strings of data type STRING2 and STRINGN shall be based on ISO 10646.

The declaration of a variable of type SHORT\_STRING is equivalent to declaring a structured data type for the variable which allocates a USINT variable containing the current size of the string in characters and an array of 8-bit octet elements. Character strings of this data type shall be based on ISO-8859-1.

The declaration of a variable of type STRINGI is equivalent to declaring a structured data type for the variable which allocates a USINT variable containing the number of internationalized character strings and an array of structures containing the internationalized character strings. The international character string structure contains a USINT indicating the first ASCII character of the ISO 639-2/T language, a USINT indicating the second character of the ISO 639-2/T language, a USINT indicating the third character of the ISO 639-2/T language, an EPATH (limited to the values 0xD0, 0xD5, 0xD9, and 0xDA) indicating the structure of the character string, a UINT indicating the character set which the character string is based on , and an array of 8-bit octet elements which is the actual international character string. The three characters for the language come from ISO 639-2/T (Alpha-3 Terminology Code), and the character set values come from IANA MIB Printer Codes (RFC 1759). The character set values are limited to those values that are provided in Table C-2.3. Some common language codes are shown in Table C-2.2.

SHORT\_STRING, STRING2, STRINGN and STRINGI are extensions to IEC 1131-3.

**Table C-2.2 Common Language Codes from ISO 639-2/T**

Language	First Character (language1)	Second Character (language2)	Third Character (language3)
English	e	n	g
French	f	r	a
Spanish	s	p	a
Italian	i	t	a
German	d	e	u
Japanese	j	p	n
Portuguese	p	o	r
Chinese	z	h	o
Russian	r	u	s

**Table C-2.3 Valid IANA MIB Printer Codes for Character Set Selection**

Character Set	Value
ISO-8859-1:1987	4
ISO-8859-2:1987	5
ISO-8859-3:1988	6
ISO-8859-4:1988	7
ISO-8859-5:1988	8
ISO-8859-6:1987	9
ISO-8859-7:1987	10
ISO-8859-8:1989	11
ISO-8859-9:1989	12
ISO-10646-UCS-2	1000
ISO-10646-UCS-4	1001

### **C-2.1.3 Structured Bit String Types**

Structured bit string types are a CIP extension of the derived data type mechanisms specified in subclause 2.2.3 of IEC 1131-3. These data types are based on the IEC standard bit string types (ANY\_BIT = BYTE, WORD, DWORD, or LWORD).

### **C-2.1.4 Derived Data Types**

The derived data types specified by CIP are:

- directly derived
- enumerated
- subrange
- structured
- array

These data types are defined in subclauses 1.3 and 2.3.3 of IEC 1131-3. The means of specifying these data types and their default initial values is defined in subclauses 2.3.3.1 and 2.3.3.2 of IEC 1131-3. The usage of variables of these data types is defined in subclause 2.3.3.3 of IEC 1131-3.

## **C-3 IEC 1131-3 Compliance**

Subclause 1.5.1 of IEC 1131-3 defines the requirements, which are met by programmable controller systems claiming compliance to the language standard. This provides the data type-related information to be included in the documentation of CIP functional units, which support the data types defined in this CIP standard. Subsets or extensions of this documentation are provided as appropriate to the specific compliant functional unit.

This section provides information with respect to only the data types and associated operations defined in this CIP standard. For full compliance with IEC 1131-3, documentation of the additional language elements supported by the functional unit must be provided as prescribed in subclause 1.5.1 of IEC 1131-3.

Included in this section are the following:

- Compliance Statement
- Implementation-Dependent Parameters
- Error Conditions
- Language Extensions
- Formal Syntax

### **C-3.1 Compliance Statement**

This system complies with the requirements of IEC 1131-3 for the following language features:

**Table C-3.1 Common Elements**

Table/ Feature	Feature description
10/1	BOOL data type
10/2	SINT data type
10/3	INT data type
10/4	DINT data type

Table/ Feature	Feature description
10/5	LINT data type
10/6	USINT data type
10/7	UINT data type
10/8	UDINT data type
10/9	ULINT data type
10/10	REAL data type
10/11	LREAL data type
10/12	TIME data type
10/13	DATE data type
10/14	TIME_OF_DAY or TOD data type
10/15	DATE_AND_TIME or DT data type
10/16	STRING data type
10/17	BYTE data type
10/18	WORD data type
10/19	DWORD data type
10/20	LWORD data type
12/1	Directly derived data types
12/2	Enumerated data types
12/3	Subrange data types
12/4	Array data types
12/5	Structured data types
13	Standard default initial values
Subclause 2.5.1.3	User-declared functions (no table entry)
22/1	Type conversions (see Table C-3.3)
22/2	TRUNC function
22/3	BCD_TO_** functions
22/4	*_TO_BCD functions
23/1-11	Standard functions of one numeric variable: ABS, SQRT, LN, LOG, EXP, SIN, COS, TAN, ASIN, ACOS, ATAN
24/12n-18n	Standard named arithmetic functions: ADD, MUL, SUB, DIV, MOD, EXPT, MOVE
24/12s-15s, 17s, 18s	Standard symbolic arithmetic functions: +, *, -, /, **, :=
25/1-4	Standard bit string functions: SHL, SHR, ROR, ROL
26/5s-8s	Standard named bitwise Boolean functions: AND, OR, XOR, NOT
26/5s-7s	Standard symbolic bitwise Boolean functions: &, >=1, =2k+1
27/1-4	Standard selection functions: SEL, MAX, MIN, LIMIT, MUX
28/5n-10n	Standard named comparison functions: GT, GE, EQ, LE, LT, NE
28/5s-10s	Standard symbolic comparison functions: >, >=, =, <=, <, <>
29/1-9	Standard character string functions: LEN, LEFT, RIGHT, MID, CONCAT, INSERT, DELETE, REPLACE, FIND
30/1-14	Standard functions of time data types: ADD, SUB, MUL, DIV, CONCAT, DATE_AND_TIME_TO_TIME_OF_DAY, TIME_OF_DAY_TO_DATE_AND_TIME NOTE— Table 30 of IEC 1131-3 limits the data types to which these operations apply.
31/1-4	Standard functions of enumerated data types: SEL, MUX, EQ, NE
32	Standard access mechanisms to function block I/O parameters
33/8a, 8b, 9a, 9b	User-defined function blocks per subclause 2.5.2.2, with graphical or textual rising or falling edge input options
34/1-3	Standard bistable function blocks: SR, RS, SEMA
35/1,2	Standard edge detection function blocks: R_EDGE, F_EDGE
36/1-3	Standard counter function blocks: CTU, CTD, CTUD

Table/ Feature	Feature description
37/1,2a, 3a, 4	Standard timer function blocks: TP, TON, TOF, RTC
55/1-17	Standard operators: (), function evaluation, **, -, NOT, *, /, MOD, +, -, <, >, <=, >=, =, <>, &, AND, XOR, OR

**Table C-3.2 ST Language Elements**

Table/ Feature	Feature description
55/1-17	Standard operators: (), function evaluation, **, -, NOT, *, /, MOD, +, -, <, >, <=, >=, =, <>, &, AND, XOR, OR
56/2	Function block invocation and output usage

**Table C-3.3 Type Conversion Operations**

Refer to the notes following this table.

Operation	Result	Error conditions
ANY_BIT_TO_ANY_BIT	(NOTE 4)	None
ANY_BIT_TO_ANY_INT	OUT <sub>min</sub> + S <sub>b</sub> <sub>k</sub> 2 <sup>k</sup> (NOTE 5)	Result > OUT <sub>max</sub>
ANY_BIT_TO_BOOL	FALSE if IN = 0 TRUE otherwise	None
ANY_BIT_TO_STRING	(NOTE 6)	None
ANY_DATE_TO_STRING	(NOTE 7)	None
ANY_INT_TO_BOOL	FALSE if IN = 0 TRUE otherwise	None
ANY_NUM_TO_ANY_INT	IN (NOTE 8)	(IN > OUT <sub>max</sub> ) or (IN < OUT <sub>min</sub> )
ANY_NUM_TO_ANY_REAL	IN	(NOTE 9)
ANY_NUM_TO_DATE	(NOTE 10)	
ANY_NUM_TO_STRING	(NOTE 11)	
ANY_NUM_TO_TIME	(NOTE 12)	
ANY_NUM_TO_TOD	(NOTE 13)	
ANY_REAL_TO_BOOL	FALSE if IN = 0.0 TRUE otherwise	None
BOOL_TO_ANY_BIT	0 if IN = FALSE 1 if IN = TRUE	None
BOOL_TO_ANY_REAL	0.0 if IN = FALSE 1.0 if IN = TRUE	None
BOOL_TO_STRING	'FALSE' if IN = FALSE 'TRUE' if IN = TRUE	None
DATE_TO_ANY_NUM	(NOTE 14)	Result > OUT <sub>max</sub>
STRING_TO_ANY	Converted data	(NOTE 15)

Operation	Result	Error conditions
TIME_TO_ANY_NUM	(NOTE 16)	Result > OUT <sub>max</sub>
TOD_TO_ANY_NUM	(NOTE 17)	Result > OUT <sub>max</sub>
1	Use of the generic data types ANY_NUM, ANY_REAL, ANY_INT, and ANY_BIT defined in subclause 2.3.2 of IEC 1131-3 is intended to imply a family of conversions. For instance, the conversion BOOL_TO_ANY_REAL is intended to imply BOOL_TO_REAL and BOOL_TO_LREAL.	
2	IN refers to the value of the input variable of the type conversion function.	
3	OUT <sub>min</sub> and OUT <sub>max</sub> refer to the minimum and maximum values of the output data type of the conversion function, as defined in Table C-2.1.	
4	In conversions of bit string types, if the number of bits in the input variable IN is less than the number of the bits in the output variable OUT, the bits of the input is copied to the corresponding least significant bits of the result and the remainder of the result is zero-filled. If the number of bits of IN is greater than the number of bits of OUT, the least significant bits of IN is copied to the corresponding bits of the result. For instance, BYTE_TO_WORD(16#FF) = 16#00FF and WORD_TO_BYTE(16#0FF0) = 16#F0	
5	Bit numbering in this expression is as specified in Note 11 of Table C-2.1.	
6	The result of conversion of a bit string variable to type STRING consists of a string containing the base 16 literal representation of the variable value, as defined in subclause 2.2.1 of IEC 1131-3, in characters taken from the ISO 646 character set.	
7	The result of conversion of a date and/or time of day variable to type STRING consists of a string containing the literal representation of the variable value, as defined in subclause 2.2.1 of IEC 1131-3, in characters taken from the ISO 646 character set.	
8	Conversion of REAL and LREAL to integer types is accomplished by rounding as specified in subclause 5.4 of IEEE 754.	
9	Rounding errors may occur if the number of significant bits in the input variable is larger than the number of significant bits in the output floating-point representation. Also, range errors of the type noted for ANY_NUM_TO_ANY_INT may occur in LREAL_TO_REAL.	
10	Conversion of a variable of a numeric type to DATE has the same result as conversion of the variable to UINT, with the result being interpreted as the number of days since 1972-01-01.	
11	Conversion of a variable of a numeric type to type STRING consists of a string containing the literal representation of the variable value, as defined in subclause 2.2.1 of IEC 1131-3, in characters taken from the ISO 646 character set.	
12	Conversion of a variable of a numeric type to TIME has the same result as conversion of the variable to DINT, with the result being interpreted as a duration in milliseconds.	
13	Conversion of a variable of a numeric type to TOD has the same result as conversion of the variable to UDINT, with the result being interpreted as a time since midnight in milliseconds.	
14	Conversion of a variable of type DATE to a numerical type is the same as the conversion of a variable of type UINT to the corresponding numerical type, with the result being the numerical equivalent of the days since 1972-01-01.	
15	It is an error if the STRING data to be converted is not in the format for external representation of the output data type as specified in subclause 2.2 of IEC 1131-3, or if the result of the conversion is outside the range {OUT <sub>min</sub> :OUT <sub>max</sub> }.	
16	Conversion of a variable of type TIME to a numerical type is the same as the conversion of a variable of type DINT to the corresponding numerical type, with the result being the numerical equivalent of the corresponding time interval expressed in milliseconds.	
17	Conversion of a variable of type TOD (TIME_OF_DAY) to a numerical type is the same as the conversion of a variable of type UDINT to the corresponding numerical type, with the result being the numerical equivalent of the time since midnight expressed in milliseconds.	

## C-3.2 Implementation-Dependent Parameters

Table C-3.4 lists the values of implementation-dependent parameters from Table D.1 of IEC 1131-3 that are defined in this CIP standard. Values of other implementation-dependent parameters are defined in other CIP standards or in the specifications of individual functional units as appropriate.

**Table C-3.4 Values of Implementation–dependent Parameters**

Clause of IEC 1131-3	Parameter	Value
2.2.3.1	Range of values of duration	Same as LINT in microseconds
2.3.1	Range of values for variables of type TIME	Same as DINT in milliseconds
	Precision of representation of seconds in types TIME_OF_DAY and DATE_AND_TIME	1 millisecond
2.3.3.1	Maximum number of enumerated values	256
2.3.3.2	Default maximum length of STRING variables	256
	Maximum allowed length of STRING variables	65536
2.4.1.2	Maximum number of subscripts	8
	Maximum range of subscript values	0..255
	Maximum number of levels of structures	8
2.5.1.5	Maximum inputs of extensible functions	8
2.5.1.5.1	Effects of type conversions on accuracy	As defined in Table C-3.3
2.5.1.5.2	Accuracy of functions of one variable	As defined in IEEE 754
2.5.2.3.3	PVmin, PVmax of counters	0, 65535

## C-3.3 Language Extensions

The extensions to IEC 1131-3 defined in this CIP standard are listed in Table C-3.5. When these extensions are used in a particular CIP compliant functional unit, the subclause references in this table are replaced by references to the descriptions of the corresponding extensions in the functional unit's documentation.

**Table C-3.5 CIP standard extensions to IEC 1131-3**

Subclause	Description
2.1.1	ITIME data type FTIME data type LTIME data type STRING2 data type STRINGN data type SHORT_STRING data type STRINGI data type EPATH data type ENGUNIT data type
2.1.4	Structured bit string types
2.2	Operations on STRING2 variables
2.2.4.1	Numbered bit string access
2.2.4.2	Structured bit string access

## C-4 Abstract Syntax Specification

The lower layers of open system architectures are concerned with the transport of user data among distributed functional units. In these layers, the user data can be regarded simply as a sequence of octets. However, application layer entities may manipulate the values of quite complex data types. To achieve independence between the application layer and lower layers, data types are specified in an abstract syntax notation.

Supplementing the abstract syntax with one or more algorithms (called encoding rules) determines the values of the lower layer octets which carry the application layer values. The combination of the abstract syntax with a single set of transfer rules produces a specific transfer syntax.

**Important:** Users of this CIP standard should read and understand ISO 8824/8825, IEC 1131-3, and section C-4, Abstract Syntax Specification. Refer to these documents when reading and applying this standard.

The data type definitions provided in this CIP standard are written in Abstract Syntax Notation One (ASN.1), as defined in ISO 8824. These type definitions are part of the ASN.1 module “CIPDataTypes.” The beginning ASN.1 statement indicating that these definitions are in this module is:

```
CIPDataTypes DEFINITIONS ::= BEGIN
```

and the closing ASN.1 statement is the keyword “END”.

The abstract definitions that follow comprise the set of CIP Data Types. In addition, provision is made to extend or derive new data types based on existing defined types, and to include those in a “type dictionary.”

### C-4.1 CIP Data Specification

The notation [typeId] for directly derived, enumerated, subrange and structured bit string data means that the tag is to be taken from the “type” field in the corresponding VariableDictionaryEntry.

```
CIPData           ::= CHOICE{ElementaryData, DerivedData}
ElementaryData    ::= CHOICE{
    BOOL,
    FixedLengthInteger,
    FixedLengthReal,
    AnyTime,
    AnyDate,
    AnyString,
    FixedLengthBitString,
    EPATH,
    ENGUNIT}
DerivedData      ::= CHOICE {
    DirectlyDerivedData,
    EnumeratedData,
    SubrangeData,
    StructuredBitStringData,
```

```

ARRAY,
STRUCT,
FunctionBlockData}

DirectlyDerivedData      ::= [ typeId ] CIPData
EnumeratedData          ::= [ typeId ] USINT
SubrangeData             ::= [ typeId ] FixedLengthInteger
StructuredBitStringData ::= [ typeId ] FixedLengthBitString
FixedLengthInteger       ::= CHOICE { SignedInteger,
                                         UnsignedInteger }

SignedInteger            ::= CHOICE { SINT, INT, DINT, LINT }
UnsignedInteger          ::= CHOICE { USINT, UINT, UDINT, ULINT }
FixedLengthReal           ::= CHOICE { REAL, LREAL }

AnyTime     ::= CHOICE { ITIME, TIME, FTIME, LTIME }
AnyDate     ::= CHOICE { DATE, TIME_OF_DAY, DATE_AND_TIME }
AnyString    ::= CHOICE { STRING, STRING2 }

FixedLengthBitString     ::= CHOICE { BYTE, WORD, DWORD, LWORD }

BOOL        ::= [ PRIVATE 1 ] IMPLICIT BOOLEAN
SINT         ::= [ PRIVATE 2 ] IMPLICIT OCTET STRING-- 1 octet
INT          ::= [ PRIVATE 3 ] IMPLICIT OCTET STRING-- 2 octets
DINT         ::= [ PRIVATE 4 ] IMPLICIT OCTET STRING-- 4 octets
LINT         ::= [ PRIVATE 5 ] IMPLICIT OCTET STRING-- 8 octets
USINT        ::= [ PRIVATE 6 ] IMPLICIT OCTET STRING-- 1 octet
UINT          ::= [ PRIVATE 7 ] IMPLICIT OCTET STRING-- 2 octets
UDINT        ::= [ PRIVATE 8 ] IMPLICIT OCTET STRING-- 4 octets
ULINT        ::= [ PRIVATE 9 ] IMPLICIT OCTET STRING-- 8 octets
REAL          ::= [ PRIVATE 10 ] IMPLICIT OCTET STRING-- 4 octets
LREAL         ::= [ PRIVATE 11 ] IMPLICIT OCTET STRING-- 8 octets
STIME         ::= [ PRIVATE 12 ] IMPLICIT DINT
DATE          ::= [ PRIVATE 13 ] IMPLICIT UINT
TIME_OF_DAY   ::= [ PRIVATE 14 ] IMPLICIT UDINT
DATE_AND_TIME ::= [ PRIVATE 15 ] IMPLICIT SEQUENCE {
    time_of_day    UDINT,
    date          UINT }

STRING        ::= [ PRIVATE 16 ] IMPLICIT SEQUENCE {
    Charcount     UINT,
    stringcontents OCTET STRING} -- one octet per character
BYTE          ::= [ PRIVATE 17 ] IMPLICIT OCTET STRING-- 1 octet
WORD          ::= [ PRIVATE 18 ] IMPLICIT OCTET STRING-- 2 octets
DWORD         ::= [ PRIVATE 19 ] IMPLICIT OCTET STRING-- 4 octets
LWORD         ::= [ PRIVATE 20 ] IMPLICIT OCTET STRING-- 8 octets
STRING2       ::= [ PRIVATE 21 ] IMPLICIT SEQUENCE {
    charcount     UINT,
    string2contents OCTET STRING} -- 2 octets/ character
FTIME         ::= [ PRIVATE 22 ] IMPLICIT DINT
LTIME          ::= [ PRIVATE 23 ] IMPLICIT LINT
ITIME          ::= [ PRIVATE 24 ] IMPLICIT INT

STRINGN       ::= [ PRIVATE 25 ] IMPLICIT SEQUENCE {
    charsize      UINT,
    charcount     UINT,

```

```

        stringNcontents OCTET STRING} -- N octets/ character
SHORT_STRING ::= [PRIVATE 26] IMPLICIT SEQUENCE {
    charcount      USINT,
    stringcontents OCTET STRING} -- one octet per character
TIME     ::= [PRIVATE 27] IMPLICIT DINT
EPATH    ::= [PRIVATE 28] IMPLICIT OCTET STRING -- Appendix C
ENGUNIT ::= [PRIVATE 29] IMPLICIT OCTET STRING -- Appendix D
STRINGI ::= [PRIVATE 30] IMPLICIT SEQUENCE{
    Stringnum    USINT (number of strings)
    array of     STRUCT
    language1   USINT (first character from ISO 639-2/T)
    language2   USINT (second character from ISO 639-2/T)
    language3   USINT (third character from ISO 639-2/T)
    datatype    EPATH limited to the values 0xD0, 0xD5, 0xD9, and 0xDA)
    charset     UINT from IANA MIB Printer Codes (RFC 1759))
    stringcontents CHOICE OF (SHORT_STRING, STRING, STRING2, or
                           STRINGN) -- based on datatype field
ARRAY    ::= SEQUENCE OF CIPData -- All of same base type
STRUCT   ::= SEQUENCE OF CIPData -- May be different types
FunctionBlockData ::= SET{
    inputs       [0] IMPLICIT STRUCT OPTIONAL,
    outputs      [1] IMPLICIT STRUCT OPTIONAL,
    controlInputs [2] IMPLICIT STRUCT OPTIONAL,
    controlOutputs [3] IMPLICIT STRUCT OPTIONAL}

```

## C-4.2 Data Type Specification/Dictionaries

The definition of a CIP object may include text that defines attributes. Attributes are assigned a *Data Type* in an object definition. The Data Type may be one of those defined in this appendix or may be an object specific extension to this appendix. The following definition provides a *Type Specification* for CIPData and provides a structure for extending or deriving new data types based on existing defined types.

```

Dictionary ::= CHOICE {VariableDictionary, TypeDictionary}
VariableDictionary ::= SEQUENCE OF VariableDictionaryEntry
VariableDictionaryEntry ::= SEQUENCE{
    name      AnyString,
    id        FixedLengthInteger,
    type      TypeID,
    ranges    SEQUENCE OF Subrange,-- for arrays
    accessPrivilege  BOOL {READ_ONLY(0), READ_WRITE(1)}
TypeID    := OCTET STRING -- ASN.1 encoded tag value of the
                     -- DataTypeSpecification module
Subrange  ::= SEQUENCE {
    minValue FixedLengthInteger,
    maxValue FixedLengthInteger}
TypeDictionary ::= SEQUENCE OF TypeDictionaryEntry
TypeDictionaryEntry ::= SEQUENCE {
    name AnyString,
    type TypeID,
    spec DataTypeSpecification}
DataTypeSpecification ::= CHOICE {
    alt      [PRIVATE 0] IMPLICIT AlternateTypeSpec,

```

```

bool      [PRIVATE 1] IMPLICIT NULL,      --  BOOL
sint      [PRIVATE 2] IMPLICIT NULL,      --  SINT
int       [PRIVATE 3] IMPLICIT NULL,      --  INT
dint      [PRIVATE 4] IMPLICIT NULL,      --  DINT
lint      [PRIVATE 5] IMPLICIT NULL,      --  LINT
usint     [PRIVATE 6] IMPLICIT NULL,      --  USINT
uint       [PRIVATE 7] IMPLICIT NULL,      --  UINT
udint     [PRIVATE 8] IMPLICIT NULL,      --  UDINT
ulint     [PRIVATE 9] IMPLICIT NULL,      --  ULINT
real      [PRIVATE 10] IMPLICIT NULL,     --  REAL
lreal     [PRIVATE 11] IMPLICIT NULL,     --  LREAL
stime     [PRIVATE 12] IMPLICIT NULL,     --  STIME
date      [PRIVATE 13] IMPLICIT NULL,     --  DATE
tod       [PRIVATE 14] IMPLICIT NULL,     --  TIME_OF_DAY
dat        [PRIVATE 15] IMPLICIT NULL,    --  DATE_AND_TIME
str1      [PRIVATE 16] IMPLICIT NULL,    --  STRING
byte      [PRIVATE 17] IMPLICIT NULL,    --  BYTE
word      [PRIVATE 18] IMPLICIT NULL,    --  WORD
dword     [PRIVATE 19] IMPLICIT NULL,    --  DWORD
lword     [PRIVATE 20] IMPLICIT NULL,    --  LWORD
str2      [PRIVATE 21] IMPLICIT NULL,    --  STRING2
ftime     [PRIVATE 22] IMPLICIT NULL,    --  FTIME
ltime     [PRIVATE 23] IMPLICIT NULL,    --  LTIME
itime      [PRIVATE 24] IMPLICIT NULL,   --  ITIME
strN      [PRIVATE 25] IMPLICIT NULL,   --  STRINGN
shstr     [PRIVATE 26] IMPLICIT NULL,   --  SHORT_STRING
time      [PRIVATE 27] IMPLICIT NULL,   --  TIME
epath     [PRIVATE 28] IMPLICIT NULL,   --  EPATH
engunit   [PRIVATE 29] IMPLICIT NULL,   --  ENGUNIT
strI      [PRIVATE 30] IMPLICIT NULL,   --  STRINGI

constructedData CHOICE {
    abbrvStruc [0] IMPLICIT AbbreviatedStrucTypeSpec,
    abbrvArr   [1] IMPLICIT AbbreviatedArrayTypeSpec,
    frmlStruc  [2] IMPLICIT FormalStrucTypeSpec,
    frmlArr    [3] IMPLICIT FormalArrayTypeSpec,
    expBitStr  [4] IMPLICIT ExpandedFixedLenBitStrTypeSpec,
    expStr1    [5] IMPLICIT ExpandedStringTypeSpec,
    expStr2    [6] IMPLICIT ExpandedString2TypeSpec }
}

AbbreviatedStrucTypeSpec      ::=  UINT
AbbreviatedArrayTypeSpec      ::=  DataTypeSpecification
FormalStrucTypeSpec  ::=  SEQUENCE OF DataTypeSpecification
FormalArrayTypeSpec  ::=  SEQUENCE {
    lowBound   [0] IMPLICIT FixedLengthInteger, -- Array Lower Bound
    highBound  [1] IMPLICIT FixedLengthInteger, -- Array Upper Bound
    dataType   DataTypeSpecification }

ExpandedFixedLenBitStrTypeSpec  ::=  SEQUENCE {
    bitStrType DataTypeSpecification -- BYTE, WORD, DWORD, or LWORD
    bitFields [7] IMPLICIT BitFieldDef }

BitFieldDef  ::=  SEQUENCE OF {
    bitDef     [2] IMPLICIT OCTET STRING}
                                -- Length is always 2 octets.
                                -- First octet contains starting
                                -- Bit Position. Trailing octet
                                -- contains the number of bits.

ExpandedStringTypeSpec  ::=  UINT-- String Length In Octets

```

```
ExpandedString2TypeSpec ::= UINT-- String Length In Octets
AlternateTypeSpec ::= CHOICE {
    directlyDerivedTypeSpec [0] IMPLICIT TypeID,
    subrangeTypeSpec [1] IMPLICIT SubrangeTypeSpec,
    enumeratedTypeSpec [2] IMPLICIT enumeratedTypeSpec,
    fbTypeSpec [3] IMPLICIT FBTypeSpec}
SubrangeTypeSpec ::= SEQUENCE{
    baseType TypeID, -- NOTE: minValue and maxValue
    minValue FixedLengthInteger, -- must be within the range
    maxValue FixedLengthInteger} -- of baseType values
EnumeratedTypeSpec ::= SEQUENCE OF AnyString
BitNameDefintion ::= SEQUENCE {
    bitName AnyString,
    bitNumber USINT}
FBTypeSpec ::= SET{
    inputs [0] IMPLICIT FbtElementSpec OPTIONAL,
    outputs [1] IMPLICIT FbtElementTypeSpec OPTIONAL,
    controlInputs [2] IMPLICIT FbtElementSpec OPTIONAL,
    controlOutputs [3] IMPLICIT FbtElementTypeSpec OPTIONAL}
FbtElementTypeSpec ::= SEQUENCE OF ElementSpec
ElementSpec ::= SEQUENCE {
    name AnyString,
    typespec ElementTypeSpec}
ElementTypeSpec ::= CHOICE {
    [0] IMPLICIT TypeID,
    [1] IMPLICIT SubrangeTypeSpec,
    [2] IMPLICIT EnumeratedTypeSpec,
    [3] IMPLICIT FormalArrayTypeSpec,
    [4] IMPLICIT ExpandedStringTypeSpec,
    [5] IMPLICIT ExpandedString2TypeSpec}
```

Additional elements of “FBTypeSpec” are being considered.

The following END statement terminates the ASN.1 module opened in section C-4.

END .

## C-5 CIP Application Transfer Syntax: Compact Encoding

This section describes the means by which the data types defined in section C-2.1 Data Type Values are encoded/transferred across CIP. The abstract syntax definition along with a particular set of encoding rules results in a transfer syntax. For CIP application user data, a single set of encoding rules is defined (Compact Encoding), resulting in the Compact transfer syntax.

Compact Encoding rules start with the encoding rules defined in ASN.1 8825. Compact Encoding then applies optimization rules, starting with the outer most Service Data Unit (SDU) and progressing to each successive encapsulated SDU. Compact Encoding defines a more efficient encoding mechanism by reducing the amount of information (overhead) transferred between devices.

The difference between a Compact encoded value and an ASN.1 encoded value is the removal of the fields describing the type and length of the information. In other words, the TAG and LENGTH components of an ASN.1-encoded value are not transmitted on CIP. In addition, the Compact Encoding rules indicate that octet-ordering rules are the reverse of those seen in ASN.1.

Given the conditions listed in the next section, the following general rules are applied to an ASN.1 encoded value to generate a Compact encoded value:

- Remove the Identifier Octets. Remove the “TAG” octets specified by ASN.1.
- Remove the Length Octets. Remove the “LENGTH” octets specified by ASN.1.
- Reverse the byte ordering for multiple content octets.

### C-5.1 Compact Encoding Constraints

**Important:** The representation of a variable value using Compact Encoding is only possible with the following restrictions:

- In a multi-peer communication relationship, the entities involved in the pre-established connection have prior knowledge of the variable type, and do not need to transmit the type description with the value of the variable. This knowledge is available by accessing the description of the variable and the variable type.
- The variable type is fixed length and has no conditional or optional fields.
- The encoding of a given variable is represented with a constant number of octets derived from the type specification of this variable.

### C-5.2 Encoding Rules/Examples

This section lists rules specific to the various data types implemented in the CIP system and shows examples of the Compact Encoding.

#### C-5.2.1 BOOL Encoding

The boolean encoding is performed on a single OCTET.

**Table C-5.1 Boolean Encoding**

If the value is:	Then:
FALSE	bit 0 of the octet is 0 ('00'H)
TRUE	bit 0 of the octet is 1 ('01'H)

The example illustrated below depicts the encoding of a BOOL whose value is FALSE.

**Table C-5.2 Example Compact Encoding of a BOOL Value**

Octet Number	1st
BOOL	00

#### C-5.2.2 SignedInteger Encoding

This section gives you examples of the Compact Encoding of SINT, INT, DINT, LINT data values. A generic illustration of the encoding of SignedInteger values is given below:

**Table C-5.3 SignedInteger Encoding**

Octet Number	1st	2nd	3rd	4th	5th	6th	7th	8th
SINT	0LSB							
INT	0LSB	1LSB						
DINT	0LSB	1LSB	2LSB	3LSB				
LINT	0LSB	1LSB	2LSB	3LSB	4LSB	5LSB	6LSB	7LSB

The following example illustrates the encoding of a variable of type DINT whose value is 12345678<sub>hex</sub>.

**Table C-5.4 Example Compact Encoding of a SignedInteger Value**

Octet Number	1st	2nd	3rd	4th
DINT	78	56	34	12

### C-5.2.3 UnsignedInteger Encoding

This section gives you examples of the Compact Encoding of USINT, UINT, UDINT, ULINT, and ENGUNIT data values. A generic illustration of the encoding of UnsignedInteger values is given below:

**Table C-5.5 UnsignedInteger Encoding**

Octet Number	1 <sup>st</sup>	2 <sup>nd</sup>	3 <sup>rd</sup>	4 <sup>th</sup>	5 <sup>th</sup>	6 <sup>th</sup>	7 <sup>th</sup>	8 <sup>th</sup>
USINT	0LSB							
UINT	0LSB	1LSB						
UDINT	0LSB	1LSB	2LSB	3LSB				
ULINT	0LSB	1LSB	2LSB	3LSB	4LSB	5LSB	6LSB	7LSB
ENGUNIT	0LSB	1LSB						

The example below illustrates the encoding of a variable of type UDINT whose value is AABBCCDD<sub>hex</sub>.

**Table C-5.6 Example Compact Encoding of a UnsignedInteger Value**

Octet Number	1st	2nd	3rd	4th
UDINT	DD	CC	BB	AA

### C-5.2.4 FixedLengthReal Encoding

This section gives you examples of the Compact Encoding of REAL and LREAL data values. A generic illustration of the encoding of FixedLengthReal values is given below:

**Table C-5.7 FixedLengthReal Encoding**

Octet Number	1st	2nd	3rd	4th	5th	6th	7th	8th
REAL	0LSB	1LSB	2LSB	3LSB				
LREAL	0LSB	1LSB	2LSB	3LSB	4LSB	5LSB	6LSB	7LSB

The example below illustrates the encoding of a variable (Float1) whose type is REAL and whose value is Float1: = 10.0. (The assignment of the value is using the IEC 1131-3 notation. The ASN.1 value is {’41200000’H} in IEEE format ( $1.25 \times 2^3$ , exponent is 130(bias 127), fraction is 25)).

**Table C-5.8 Example Compact Encoding of a REAL Value**

Octet Contents	0LSB	1LSB	2LSB	3LSB
REAL	00	00	20	41

The example below illustrates the encoding of a variable (Float2) whose type is LREAL and whose value is Float2: = -100.0. ( {`C059000000000000'H} in IEEE format ( $1.5625 \times 2^6$ , exponent is 1029 (bias 1023), fraction is .5625)).

**Table C-5.9 Example Compact Encoding of a LREAL Value**

Octet Contents	0LSB	1LSB	2LSB	3LSB	4LSB	5LSB	6LSB	7LSB
LREAL	00	00	00	00	00	00	59	C0

### C-5.2.5 Time Encodings

This section gives you examples of the Compact Encoding of TIME, DATE, TIME\_OF\_DAY, DATE\_AND\_TIME, FTIME, LTIME, ITIME data values. A generic illustration of the encoding of Time values are given below:

**Table C-5.10 Time Encoding**

Octet Number	1st	2nd	3rd	4th	5th	6th	7th	8th
TIME	0LSB	1LSB	2LSB	3LSB				
DATE	0LSB	1LSB						
TIME_OF_DAY	0LSB	1LSB	2LSB	3LSB				
DATE_AND_TIME	0LSB- Time	1LSB- Time	2LSB- Time	3LSB- Time	0LSB- Date	1LSB- Date		
FTIME	0LSB	1LSB	2LSB	3LSB				
LTIME	0LSB	1LSB	2LSB	3LSB	4LSB	5LSB	6LSB	7LSB

### C-5.2.6 String Encodings

This section gives you examples of the Compact Encoding of STRING, STRING2, STRINGN, and SHORT\_STRING data values.

**Important:** The preferred string type for user supplied string data is STRING2 due to international character string requirements.

A generic illustration of the encoding of a STRING value is given below:

**Table C-5.11 STRING Encoding**

	Contents (charcount)		Contents (string contents)
STRING	0LSB	1LSB	0LSB
	1 byte character		

A generic illustration of the encoding of a STRING2 value is given below:

**Table C-5.12 STRING2 Encoding**

	Contents (charcount)		Contents (string2contents)	
STRING2	0LSB	1LSB	0LSB	1LSB
	2 byte character			

A generic illustration of the encoding of a STRINGN value is given below:

**Table C-5.13 STRINGN Encoding**

	Contents (charsize)		Contents (charcount)		Contents (stringNcontents)	
STRINGN	0LSB	.....	1LSB	0LSB	.....	1LSB

0LSB ..... NLSB  
N byte character

A generic illustration of the encoding of a SHORT\_STRING value is given below:

**Table C-5.14 SHORT\_STRING Encoding**

	Contents (charcount)	Contents (short_string)
SHORT_STRING	0LSB	0LSB 1 byte character

The example below illustrates the encoding of a string variable whose contents equal "Mill". Encoding examples of all string types are presented. Character coding is specified in ISO 10646. The hexadecimal equivalent is: { '4D696C6C'H} for 8 bit encoding.

The example below encodes "Mill" as a STRING type.

**Table C-5.15 Example Compact Encoding of A String Value**

	Contents (charcount)		Contents (string contents)				
STRING	04	00	4D	69	6C	6C	

The example below encodes "Mill" as a STRING2 type.

**Table C-5.16 Example Compact Encoding of String2 Value**

	Contents (charcount)		Contents (string2 contents)							
STRING2	04	00	4D	00	69	00	6C	00	6C	00

2 byte character

The example below encodes "Mill" as a SHORT\_STRING type.

**Table C-5.17 Example Compact Encoding of SHORT\_STRING Value**

	Contents (charcount)	Contents (short_string contents)				
SHORT_STRING	04	4D	69	6C	6C	

### C-5.2.7 FixedLengthBitString Encoding

This section supplies examples of the Compact Encoding of BYTE, WORD, DWORD, LWORD data values. The table below illustrates the bit placement rules associated with the Compact Encoding of a FixedLengthBitString.

**Table C-5.18 Example Compact Encoding of a FixedLengthBitString Value**

Octet Number→	1st	2nd	3rd	4th	5th	6th	7th	8th
BYTE	7.....0							
WORD	7.....0	15.....8						
DWORD	7.....0	15.....8	23.....16	31.....24				
LWORD	7.....0	15.....8	23.....16	31.....24	39.....32	47.....40	55.....48	63.....56

The examples below illustrate the encoding of a BYTE, WORD, DWORD, and an LWORD.

**Figure C-5.1 Example Compact Encoding of A BYTE FixedLengthBitString**

Bits In Memory:  
7 ..... 0  
00001111

Compact Encoded BYTE  
00001111 or 0Fhex

**Figure C-5.2 Example Compact Encoding of A WORD FixedLengthBitString**

Bits In Memory:  
15 ..... 0  
00001111 11001111

Compact Encoded WORD  
11001111 00001111 or CF0hex

**Figure C-5.3 Example Compact Encoding of A DWORD FixedLengthBitString**

Bits In Memory:  
31 ..... 0  
00001111 11001111 10110110 00111110

Compact Encoded DWORD  
00111110 10110110 11001111 00001111 or 3EB6CF0hex

**Figure C-5.4 Example Compact Encoding of A LWORD FixedLengthBitString**

Bits In Memory:  
63 ..... 0  
11110000 11001111 10110110 00111110 11110000 00101101 00011110 00001111

Compact Encoded LWORD  
00001111 00011110 00101101 11110000 00111110 10110110 110011111110000 or 0F1E2DF03EB6CFF0hex

### C-5.2.8 Array Encoding

The array encoding uses the encoding rules for the CIP Data types for each element and concatenates the elements which compose the array. The encoded values of the array elements are encoded in the same order as they are declared in the corresponding ASN.1 type or variable specification. These elements may be of any CIP Data type.

Array definitions follow FIP and FieldBus standards, whose committees are currently planning to specify the bounding limits for each dimension of an array. The ASN.1-style definition of a single-dimensional array in a CIP network is:

```
ARRAY ::= SEQUENCE OF { array_dimension_low_bound,  
                        array_dimension_high_bound, CIPData }
```

Assume the following array definition::

```
ARRAY1 ::= SEQUENCE OF {array_dimension_low_bound := 0,  
                        array_dimension_high_bound := 1, UINT}
```

Plugging the UINT values {1,2} into this array definition yields the following encoding:

**Table C-5.19 Example Compact Encoding of a Single Dimensional ARRAY**

Octet Number	1st	2nd	3rd	4th
ARRAY	01	00	02	00

The ASN.1-style definition of a two-dimensional array in a CIP network is:

```
ARRAY ::= SEQUENCE OF { array_dimension_low_bound,  
                        array_dimension_high_bound,  
                        SEQUENCE OF { array_dimension_low_bound,  
                                      array_dimension_high_bound, CIPData } }
```

The ASN.1-style definition of a three-dimensional array in a CIP network is:

```
ARRAY ::= SEQUENCE OF { array_dimension_low_bound,  
                        array_dimension_high_bound,  
                        SEQUENCE OF { array_dimension_low_bound,  
                                      array_dimension_high_bound,  
                                      SEQUENCE OF { array_dimension_low_bound,  
                                                    array_dimension_high_bound, CIPData } } }
```

Since CIPData may comprise either ElementaryData or DerivedData, a new type or variable specification may be required before transmitting the values for the ARRAY.

A multi-dimensional array is seen on the wire as a single-dimensional array. The order of the array elements is maintained via the packing/unpacking sequence followed by the end nodes. The sequence followed is to access the Nth dimension of the array for all values of the other dimensions.

This is achieved by first accessing the array with all dimensions set to their initial index values. After this the Nth dimension is incremented through all of its index values. When the end of the index range for the Nth dimension is reached, the (N-1)th dimension is incremented, and the Nth dimension is set to its initial index value. This process is repeated until all of the array's dimensions have reached the ends of their index ranges, and results in the array being packed into the message buffer as a single-dimensional array. The same procedure is followed to unpack the single-dimensional array into a multi-dimensional array.

The two-dimensional array shown results in the data stream below when it is packed into a single-dimensional array following the compact encoding rules:

```
ARRAY1 [0..1 , 0..2] of UINT := { { 1, 2, 3 }, { 4, 5, 6 } }
```

**Table C-5.20 Example Compact Encoding of a Multi-Dimensional ARRAY**

Octet Number	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th	11th	12th
ARRAY	01	00	02	00	03	00	04	00	05	00	06	00

### C-5.2.9 Structure Encoding

The structure encoding uses the encoding rules for the CIPData types for each element and concatenates the elements which compose the structure.

The encoded values of the structure elements are encoded in the same order as they are declared in the corresponding ASN.1 type or variable specification. These elements may be of any CIPData type.

```
STRUCT ::= SEQUENCE OF CIPData -- May be different types
```

Since CIPData may be comprised of either ElementaryData or DerivedData, a new type or variable specification may be required before transmitting the values for the STRUCT.

Assume the following structure definition:

```
newStruct ::= SEQUENCE { BOOL,UINT,DINT }
```

Plugging the values {TRUE,’1234’H,’56789ABC’H} into the structure results in the following encoding:

**Table C-5.21 Example Compact Encoding of a STRUCTURE**

Octet Number	1st	2nd	3rd	4th	5th	6th	7th
newStruct	01	34	12	BC	9A	78	56

### C-5.2.10 Examples of How More Complex Data Formats are Encoded

The examples below show how more complex data formats are packed. Example 1 shows the packing of an array of structures. Example 2 shows how a structure with an array element is packed.

Example 1: Encoding an Array Of Structures:

```
STRUCT1 ::= SEQUENCE OF {
    UINT     ele1;
    USINT    ele2;
    USINT    ele3;
    USINT    ele4;
    UINT     ele5
}
```

```
ARRAY1 [ 0..1 , 0..2 ] of STRUCT1 := {
{ { 1, 2, 3, 4, 5 }, { 6, 7, 8, 9, 10 } },
{ { 11, 12, 13, 14, 15 } },
{ { 15, 14, 13, 12, 11 }, { 10, 9, 8, 7, 6 } },
{ 5, 4, 3, 2, 1 } }
```

results in the following data stream:

```
[01] [00] [02] [03] [04] [05] [00] [06] [00] [07] [08] [09] [0A] [00]
[0B] [00] [0C] [0D] [0E] [0F] [00] [0F] [00] [0E] [0D] [0C] [0B] [00]
[0A] [00] [09] [08] [07] [06] [00] [05] [00] [04] [03] [02] [01] [00]
```

#### Example 2: Encoding a Structure with an Array Element

```
STRUCT2 ::= SEQUENCE OF {
    UINT      ele1;
    ARRAY [ 0..2 ] of USINT array2;
    UINT      ele5;
}
STRUCT2 := { 1, { 2, 3, 4 }, 5 }
```

results in the following data stream:

```
[01] [00] [02] [03] [04] [05] [00]
```

## C-6 Data Type Reporting

Objects may choose to implement a mechanism for reporting Data Type of a particular Attribute or transmitting type information along with the actual data. This section defines the means by which Data Typing information is conveyed.

The specification of CIP Data Type information utilizes the ASN.1 methodology specified in ISO 8824:1987(E) and ISO 8825:1987(E) with CIP defined optimizations to encode the **DataTypeSpecification** production defined in section C-4.2. The CIP defined optimizations are listed below:

- The Length Octet of a NULL type is not encoded. For example; the encoding of '*abc [PRIVATE 1] IMPLICIT NULL*' would be: C1<sub>hex</sub> (a tag with no length octet).

The sections that follow detail:

- Elementary Data Type Reporting
- Constructed Data Type Reporting

### C-6.1 Elementary Data Type Reporting

Elementary data types are identified using the identification codes defined in the table below. These codes illustrate the encoding of the primitive members of the **DataTypeSpecification** production. Remember that CIP specifies that ASN.1 NULL types do not report the Length Octet of zero (0).

**Table C-6.1 Identification Codes and Descriptions of Elementary Data Types**

Data Type Name	Data Type Code (in hex)	Data Type Description
BOOL	C1	Logical Boolean with values TRUE and FALSE

Data Type Name	Data Type Code (in hex)	Data Type Description
SINT	C2	Signed 8-bit integer value
INT	C3	Signed 16-bit integer value
DINT	C4	Signed 32-bit integer value
LINT	C5	Signed 64-bit integer value
USINT	C6	Unsigned 8-bit integer value
UINT	C7	Unsigned 16-bit integer value
UDINT	C8	Unsigned 32-bit integer value
ULINT	C9	Unsigned 64-bit integer value
REAL	CA	32-bit floating point value
LREAL	CB	64-bit floating point value
STIME	CC	Synchronous time information
DATE	CD	Date information
TIME_OF_DAY	CE	Time of day
DATE_AND_TIME	CF	Date and time of day
STRING	D0	character string (1 byte per character)
BYTE	D1	bit string - 8-bits
WORD	D2	bit string - 16-bits
DWORD	D3	bit string - 32-bits
LWORD	D4	bit string - 64-bits
STRING2	D5	character string (2 bytes per character)
FTIME	D6	Duration (high resolution)
LTIME	D7	Duration (long)
ITIME	D8	Duration (short)
STRINGN	D9	character string (N bytes per character)
SHORT_STRING	DA	character sting (1 byte per character, 1 byte length indicator)
TIME	DB	Duration (milliseconds)
EPATH	DC	CIP path segments
ENGUNIT	DD	Engineering Units
STRINGI	DE	International Character String

## C-6.2 Constructed Data Type Reporting

This section details the means by which the structure and array information presented within the **DataTypeSpecification** production is represented.

### C-6.2.1 Structure Type Definition

CIP defines two different methods for reporting Structure Type Definitions:

- Formal Encoding (**FormalStrucTypeSpec**)
- Abbreviated Encoding (**AbbreviatedStrucTypeSpec**)

Formal encoding is used to provide a detailed report of the complete structure definition, including the complete definition of all component data types. Abbreviated encoding is used to specify a *shorter* form of the structure definition. This *shorter* form does not include the data types associated with the structure's components.

### Formal Encoding for Structure Type Information

The two examples below illustrate formal encoding for structure type specifications. This is actually an example of the encoding of the **FormalStrucTypeSpec** production defined in section C-4.2, Data Type Specification/Dictionaries.

#### **Example 1:**

Table C-6.2 shows the encoding of the following structure definition

```
STRUCT ::= SEQUENCE OF { BOOL, UINT, DINT }
```

**Table C-6.2 Example 1 of Formal Encoding of a Structure Type Specification**

Struc Type	Type Length	Component Types	Component Types	Component Types
A2	03	BOOL	UINT	DINT
		C1	C7	C4

Note that the IMPLICIT NULL types from the **DataTypeSpecification** production are not followed by a Length Octet of zero (0).

#### **Example 2:**

Table C-6.3 shows the encoding of the following structure definition

```
STRUCT_MAIN ::= SEQUENCE OF { UINT, STRUCT_SUB, INT }
```

with subelement STRUCT\_SUB defined as:

```
STRUCT_SUB ::= SEQUENCE OF { UINT, SINT, INT }
```

**Table C-6.3 Example 2 of Formal Encoding of a Structure Type Specification**

Struct	Type Length	Component							
		UINT	Struct Type	Type Length	Nested Component			INT	
					UINT	SINT	INT		
A2	07	C7	A2	03	C7	C2	C3	C3	

### Abbreviated Encoding for Structure Type Information

The example below illustrates the abbreviated encoding for structure type specifications. This is actually an example of the encoding of the **AbbreviatedStrucTypeSpec** production defined in section C-4.2, Data Type Specification/Dictionaries.

The UINT defined within the AbbreviatedStrucTypeSpec production is initialized with a 16 bit Cyclic Redundancy Check (CRC) value. This can be used by application logic to determine whether or not the format of the structure has changed. The byte stream used to produce the CRC is the actual formally encoded (**FormalStrucTypeSpec**) structure type specification. See Section C-7, CRC Generation Algorithms.

**Example:**

Table C-6.4 shows the abbreviated encoding of the structure definition presented in the previous section:

**Table C-6.4 Example of Abbreviated Encoding of a Structure Type Specification**

Struc	Type Length	UINT Containing CRC	
A0	02	C7	26

Note that the algorithms presented in section used to generate the CRC value of 26C7<sub>hex</sub> from the Formally Encoded type specification: [A2][07][C7][A2][03][C7][C2][C3][C3].

### C-6.2.2 Array Type Definition

CIP defines two different methods for reporting Array Type Definitions:

- Formal Encoding (**FormalArrayTypeSpec**)
- Abbreviated Encoding (**AbbreviatedArrayTypeSpec**)

Formal encoding is used to provide a detailed report of the complete array definition, including the data content and the array's dimensions. Abbreviated encoding is used to specify a *shorter* form of the array definition. This *shorter* form does not include information specifying the array's dimensions.

#### Formal Encoding for Array Type Information

The two examples below illustrate formal encoding for structure type specifications. This is actually an example of the encoding of the **FormalArrayTypeSpec** production defined in section C-4.2, Data Type Specification/Dictionaries.

**Example 1:**

Table C-6.5 shows the formal encoding of the following array definition

```
ARRAY1 ::= SEQUENCE OF { array_dimension_low_bound := 0,
                        array_dimension_high_bound := 9,
                        UINT }
```

**Table C-6.5 Example 1 of Formal Encoding of An Array Type Specification**

Array	Type Length	Lower Bound Tag	Lower Bound Length	Lower Bound	Upper Bound Tag	Upper Bound Length	Upper Bound	UINT
A3	07	C7	01	00	81	01	09	C7

Note that the IMPLICIT NULL types from the **DataTypeSpecification** production are not followed by a Length Octet of zero (0).

**Example 2:**

Table C-6.6 shows the encoding of the following array definition

```
ARRAY1 ::= SEQUENCE OF { array_dimension_low_bound := 0,
                        array_dimension_high_bound := 19,
                        SEQUENCE OF { array_dimension_low_bound := 0,
                        array_dimension_high_bound := 255, STRUCT_ELE}
                        }
```

in which STRUCT\_ELE is defined as:

```
STRUCT_ELE ::= SEQUENCE OF { UINT, SINT, INT }
```

**Table C-6.6 Example 2 of Formal Encoding of an Array Type Specification**

Formal Encoding:

[A3] [13] [80] [01] [00] [81] [01] [13] [A3] [0B] [80] [01] [00] [81] [01] [FF] [A2] [03] [C7] [C2] [C3]

Array Type	Type Length	Lower Bound Tag	Lower Bound Length	Lower Bound	Upper Bound Tag	Upper Bound Length	Upper Bound
A3	13	80	01	00	81	01	13

...

Array Contents								
Array Type	Type Length	Lower Bound Tag	Lower Bound Length	Lower Bound	Upper Bound Tag	Upper Bound Length	Upper Bound	
A3	0B	80	01	00	81	01	FF	

...

Nested Array Contents				
Struct Type	Type Length	UINT	SINT	INT
A2	03	C7	C2	C3

Note that the IMPLICIT NULL types from the **DataTypeSpecification** production are not followed by a Length Octet of zero (0).

Abbreviated Tag Encoding for Array Type Information

The abbreviated encoding of an Array type DOES NOT include the information specifying the Array's dimensions. This is actually an example of the encoding of the **AbbreviatedArrayTypeSpec** production defined in section C-4.2, Data Type Specification/Dictionaries.

**Example 1:**

Table C-6.7 shows the abbreviated encoding of the following array definition:

```
ARRAY2 ::= SEQUENCE OF { array_dimension_low_bound := 0,
                        array_dimension_high_bound := 9,
                        UINT }
```

**Table C-6.7 Example 1 of Abbreviated Encoding of an Array Type Specification**

Array Type	Type Length	UINT
A1	01	C7

Note that the IMPLICIT NULL types from the **DataTypeSpecification** production are not followed by a Length Octet of zero (0).

**Example 2:**

Table C-6.8 shows the abbreviated encoding of the following array definition:

```
ARRAY ::= SEQUENCE OF { array_dimension_low_bound := 0,
                        array_dimension_high_bound := 19,
                        SEQUENCE OF { array_dimension_low_bound := 0,
                        array_dimension_high_bound := 899, STRUCT_ELE }
                      }
```

in which STRUCT\_ELE is defined as:

```
STRUCT_ELE ::= SEQUENCE OF { UINT, SINT, INT }
```

**Table C-6.8 Example 2 of Abbreviated Encoding of an Array Type Specification**

Array Type	Type Length	Array Contents					
		Array Type	Nested Component				
			Type Length	Struct Type	Type Length	UINT Containing CRC	
A1	06	A1	04	A0	02	59	51

## C-7

## CRC Generation Algorithms

The C routine below generates a CRC via the polynomial driven method.

```
*****
Function Name: calcPolyCrc()
Description: calculates a Cyclic Redundancy Checksum CRC) via the
polynomial driven method.
Inputs: start_addr - the address of the first byte which is involved
in the crc calculation
        size - the number of consecutive bytes that are involved in the
crc calculation. Size of 0 is invalid.
Outputs: a 16 bit value containing the calculated CRC
```

```
*****
unsigned16BitInteger calcPolyCrc(start_addr, size)
unsigned8BitInteger *start_addr;
unsigned32BitInteger size;
{
    unsigned16BitInteger crc;
    unsigned8BitInteger carry;
    unsigned8BitInteger b;
    unsigned32BitInteger cnt;
    crc = 0;
    while (size)
    {
        b = (unsigned8BitInteger) *start_addr++;
        crc ^= b;
        cnt = 0;
        while (cnt < 8)
        {
            carry = crc & 1;
            crc >>= 1;
            if (carry)
            {
                crc ^= 0xa001;
                cnt++;
            }
        }
        size--;
    }
    return (crc);
}
```

The C routine below generates a CRC via the table driven method.

unsigned16BitInteger poly_table[] = {0x0000,	0xC0C1,	0xC181,	
0x0140, 0xC301, 0x03C0,	0x0280,	0xC241,	0xC601,
0x06C0, 0x0780, 0xC741,	0x0500,	0xC5C1,	0xC481,
0x0440, 0xCC01, 0x0CC0,	0x0D80,	0xCD41,	0x0F00,
0xCFC1, 0xCE81, 0x0E40,	0x0A00,	0xCAC1,	0xCB81,
0x0B40, 0xC901, 0x09C0,	0x0880,	0xC841,	0xD801,
0x18C0, 0x1980, 0xD941,	0x1B00,	0xDBE1,	0xDA81,
0x1A40, 0x1E00, 0xDEC1,	0xDF81,	0x1F40,	0xDD01,
0x1DC0, 0x1C80, 0xDC41,	0x1400,	0xD4C1,	0xD581,
0x1540, 0xD701, 0x17C0,	0x1680,	0xD641,	0xD201,
0x12C0, 0x1380, 0xD341,	0x1100,	0xD1C1,	0xD081,
0x1040, 0xF001, 0x30C0,	0x3180,	0xF141,	0x3300,
0xF3C1, 0xF281, 0x3240,	0x3600,	0xF6C1,	0xF781,
0x3740, 0xF501, 0x35C0,	0x3480,	0xF441,	0x3C00,
0xFCC1, 0xFD81, 0x3D40,	0xFF01,	0x3FC0,	0x3E80,
0xFE41, 0xFA01, 0x3AC0,	0x3B80,	0xFB41,	0x3900,
0xF9C1, 0xF881, 0x3840,	0x2800,	0xE8C1,	0xE981,
0x2940, 0xEB01, 0x2BC0,	0x2A80,	0xEA41,	0xEE01,
0x2EC0, 0x2F80, 0xEF41,	0x2D00,	0xEDC1,	0xEC81,
0x2C40, 0xE401, 0x24C0,	0x2580,	0xE541,	0x2700,
0xE7C1, 0xE681, 0x2640,	0x2200,	0xE2C1,	0xE381,
0x2340, 0xE101, 0x21C0,	0x2080,	0xE041,	0xA001,
0x60C0, 0x6180, 0xA141,	0x6300,	0xA3C1,	0xA281,
0x6240, 0x6600, 0xA6C1,	0xA781,	0x6740,	0xA501,
0x65C0,			

```

0x6480,      0xA441,      0x6C00,      0xACC1,      0xAD81,      0x6D40,
0xAF01,      0x6FC0,      0x6E80,      0xAE41,      0xAA01,      0x6AC0,
0x6B80,      0xAB41,      0x6900,      0xA9C1,      0xA881,      0x6840,
0x7800,      0xB8C1,      0xB981,      0x7940,      0xBB01,      0x7BC0,
0x7A80,      0xBA41,      0xBE01,      0x7EC0,      0x7F80,      0xBF41,
0x7D00,      0xBD1,       0xBC81,      0x7C40,      0xB401,      0x74C0,
0x7580,      0xB541,      0x7700,      0xB7C1,      0xB681,      0x7640,
0x7200,      0xB2C1,      0xB381,      0x7340,      0xB101,      0x71C0,
0x7080,      0xB041,      0x5000,      0x90C1,      0x9181,      0x5140,
0x9301,      0x53C0,      0x5280,      0x9241,      0x9601,      0x56C0,
0x5780,      0x9741,      0x5500,      0x95C1,      0x9481,      0x5440,
0x9C01,      0x5CC0,      0x5D80,      0x9D41,      0x5F00,      0x9FC1,
0x9E81,      0x5E40,      0x5A00,      0x9AC1,      0x9B81,      0x5B40,
0x9901,      0x59C0,      0x5880,      0x9841,      0x8801,      0x48C0,
0x4980,      0x8941,      0x4B00,      0x8BC1,      0x8A81,      0x4A40,
0x4E00,      0x8EC1,      0x8F81,      0x4F40,      0x8D01,      0x4DC0,
0x4C80,      0x8C41,      0x4400,      0x84C1,      0x8581,      0x4540,
0x8701,      0x47C0,      0x4680,      0x8641,      0x8201,      0x42C0,
0x4380,      0x8341,      0x4100,      0x81C1,      0x8081,
0x4040};

//*****
FFunction Name: calcTableCrc()
Description: Calculates a 16 bit crc value based on the input parameters via
the table driven method
Inputs: start_addr - pointer to the first byte of memory involved in the
calculation of the CRC
size - the number of bytes to include in the CRC generation
Outputs: a 16 bit CRC
*****/
unsigned16BitInteger calcTableCrc(start_addr, size)
unsigned8BitInteger *start_addr;
unsigned32BitInteger size;
{
unsigned16BitInteger crc;
unsigned16BitInteger data;

crc = 0;
while (size)
{
    data = (unsigned16BitInteger) * start_addr++;
    crc = ((crc >> 8) ^ poly_table[(crc ^ data) & 255]);
    size--;
}
return (crc);
}


```

This page is intentionally left blank

## **Volume 1: Common Industrial Protocol (CIP<sup>TM</sup>) Specification**

### **Appendix D: Engineering Units**

---

## Contents

D-1	Introduction.....	3
D-1.1	Reporting Engineering Units (ENGUNIT Codes) .....	3
D-1.2	Groups of Engineering Units .....	4
D-1.3	Naming Conventions.....	5
D-2	Tables.....	6
D-2.1	Profile Specific Engineering Unit Enumerations .....	6
D-2.2	Vendor Specific Engineering Unit Enumerations .....	6
D-2.3	General.....	6
D-2.4	Time (includes Date).....	7
D-2.5	Temperature .....	7
D-2.6	Pressure .....	8
D-2.7	Flow .....	8
D-2.8	Acceleration .....	9
D-2.9	Amount of Substance .....	9
D-2.10	Angle.....	9
D-2.11	Area.....	9
D-2.12	Capacitance .....	10
D-2.13	Charge (Electric Charge).....	10
D-2.14	Conductance (Electric Conductance) .....	10
D-2.15	Current (Electric Current) .....	11
D-2.16	Energy (Heat, Work).....	11
D-2.17	Force .....	12
D-2.18	Frequency (includes RPM) <sup>#</sup> .....	12
D-2.19	Inductance .....	13
D-2.20	Inertia .....	13
D-2.21	Length (Distance, Displacement).....	13
D-2.22	Light (Luminous Intensity) .....	14
D-2.23	Magnetic Flux .....	14
D-2.24	Mass .....	14
D-2.25	Power (Wattage) .....	15
D-2.26	Radiology .....	15
D-2.27	Resistance (Electric Resistance).....	16
D-2.28	Sound .....	16
D-2.29	Torque (Moment of Force).....	16
D-2.30	Velocity (Speed) <sup>#</sup> .....	17
D-2.31	Viscosity .....	17
D-2.32	Voltage .....	17
D-2.33	Volume.....	18
D-2.34	Density .....	18

## **D-1      Introduction**

In the physical sciences, quantities are measured by comparing them to a quantity whose magnitude is defined to be unity. Such a quantity is called a *unit*. For example, a commonly used unit for mass is the kilogram.

Within the CIP specification, units are often used to describe a device's interface to the physical world. For example, a photoelectric switch might use units of length to describe its min/max detect distances, and a position controller might use units of velocity and acceleration to describe its movements.

This appendix defines a system of units for use in the CIP specification, including names, symbols, and a mechanism for reporting units. This system of units uses the commonly recognized name *engineering units*.

### **D-1.1    Reporting Engineering Units (ENGUNIT Codes)**

An attribute of data type ENGUNIT is used to specify the engineering units for one or more other attributes.

By providing a ENGUNIT attribute, an object can manage different engineering units. For example, an analog sensor object with a ENGUNIT attribute could potentially measure temperature, flow, pressure, and so on.

If the engineering unit used by an object does not vary, the unit can be specified in the object's specification, and a ENGUNIT attribute is not needed. In such cases, names and symbols of units listed in the object's specification should remain consistent with this appendix.

The ENGUNIT code is used to represent engineering units only, and does not imply a specific data type, range, or scaling. For example, if the engineering units of a UINT (16-bit unsigned integer) attribute is defined by another ENGUNIT attribute which indicates milliamps, this does **not** imply a range of 0-65535 mA. The encoding of a given unit by an object is defined entirely by that object's specification.

The ENGUNIT data type is encoded as a UINT (16-bit unsigned integer). The most significant byte of this UINT selects a group of similar units, and the least significant byte selects a specific unit within that group.

## D-1.2 Groups of Engineering Units

The following groups of engineering units are listed in the tables of section D-2:

<b>Value of MSB</b>	<b>Engineering Units Group</b>
00 hex thru 07 hex	Reserved
08 hex thru 0F hex	Vendor specific enumerations
10 hex	General
11 hex	Time (includes Date)
12 hex	Temperature
13 hex	Pressure
14 hex	Flow
15 hex	Acceleration
16 hex	Amount of Substance
17 hex	Angle
18 hex	Area
19 hex	Capacitance
1A hex	Charge (Electric Charge)
1B hex	Conductance (Electric Conductance)
1C hex	Current (Electric Current)
1D hex	Energy (Heat, Work)
1E hex	Force
1F hex	Frequency (includes RPM)
20 hex	Inductance
21 hex	Inertia
22 hex	Length (Distance, Displacement)
23 hex	Light (Luminous Intensity)
24 hex	Magnetic Flux
25 hex	Mass
26 hex	Power (Wattage)
27 hex	Radiology
28 hex	Resistance (Electric Resistance)
29 hex	Sound
2A hex	Torque (Moment of Force)
2B hex	Velocity (Speed)
2C hex	Viscosity
2D hex	Voltage
2E hex	Volume
2F hex	Density
30-FF hex	reserved for future standardization

### D-1.3 Naming Conventions

In the tables of section D-2, each unit lists:

- *Value*: Value used to represent the unit in a ENGUNIT attribute
- *Name*: Commonly used name for the unit (often used for text or display)
- *Symbol*: Commonly used symbol for the unit (often used for expressions)
- *Base Units*: Expression in terms of other base units (such as metric conversions)

Many of the names, symbols, and expressions are taken from the International System of Units (SI). <sup>#1</sup>

In addition, the following conventions are taken from SI specifications:

- The following SI prefixes are used for names and symbols in order to form decimal multiples and submultiples of units as appropriate:

Factor	Name Prefix	Symbol Prefix
$10^9$	giga	G
$10^6$	mega	M
$10^3$	kilo	k
$10^2$	hecto	h
$10^1$	deka (or deca)	da
$10^{-1}$	deci	d
$10^{-2}$	centi	c
$10^{-3}$	milli	m
$10^{-6}$	micro	$\mu$
$10^{-9}$	nano	n
$10^{-12}$	pico	p
$10^{-15}$	femto	f

- Unit symbols and unit names are not used together. For example, millivolt and mV are valid, but mvolt and milliV are not valid.
- A name (symbol) prefix is inseparable from its unit name (symbol). For example, milli amp and milli-amp are not valid names.
- A centered dot ('•') is used to indicate multiplication, and a solidus ('/') is used to indicate division.
- Each unit definition is listed in its singular form. For example, velocity is listed as meter per second, not meters per second. Plural forms are more commonly used in values for a given unit (such as 8 meters per second).
- Capitalization follows SI conventions.

Also, in order to offset footnote markers from other superscripts, each footnote marker is preceded by the # symbol.

---

<sup>1</sup> [The International System of Units \(SI\)](#), NIST Special Publication 330, United States Department of Commerce, Technology Administration, National Institute of Standards and Technology. See also: [Guide for the Use of the International System of Units \(SI\)](#), NIST Special Publication 811. These documents may be available for access on [www.nist.gov](http://www.nist.gov).

## D-2 Tables

### D-2.1 Profile Specific Engineering Unit Enumerations

Value	Name	Symbol	Base Units
0000 hex through 07FF hex	Reserved to prevent overlap with the SEMI Standard		

### D-2.2 Vendor Specific Engineering Unit Enumerations

Value	Name	Symbol	Base Units
0800 hex through 0FFF hex	This range of values is reserved for Engineering Units specific to a particular vendor's needs. Any device that utilizes values within this range must enumerate (or otherwise identify) the exact meaning of each value that can be represented. These values are reserved for previously un-enumerated Engineering Units. Engineering units that are already represented by enumerations in the following sections are not allowed to be assigned in this vendor specific range.		

### D-2.3 General

Value	Name	Symbol	Base Units
1000 hex	unspecified		
1001 hex	counts		
1002 hex	part per million (concentration)	ppm	
1003 hex	position		
1004 hex	pixel		
1005 hex	bit		
1006 hex	byte		
1007 hex	percent	%	
1008-10FF hex	reserved for future standardization		

#### D-2.4 Time (includes Date)

Value	Name	Symbol	Base Units
1100 hex	second #2	s	
1101 hex	millisecond #3	ms	$10^{-3} \bullet s$
1102 hex	microsecond #4	μs	$10^{-6} \bullet s$
1103 hex	minute	min	$60 \bullet s$
1104 hex	hour	h	$3600 \bullet s$
1105 hex	day	d	$86400 \bullet s$
1106 hex	nanosecond	ns	$10^{-9} \bullet s$
1107 hex	DATE #5	D	
1108 hex	TIME_OF_DAY #6	TOD	
1109 hex	DATE_AND_TIME #7	DT	
110A-11FF hex	reserved for future standardization		

#### D-2.5 Temperature

Value	Name	Symbol	Base Units
1200 hex	degree Celcius #8	°C	K - 273.15
1201 hex	degree Fahrenheit	°F	(K • 1.8) - 459.67
1202 hex	kelvin #9	K	
1203 hex	Degree Rankine	°R	K • 1.8
1204 hex	decidegree Celcius	°C/10	(K • 10 <sup>-1</sup> ) - 27.315
1205 hex	decidegree Fahrenheit	°F/10	(K • 1.8 • 10 <sup>-1</sup> ) - 45.967
1206 hex	decikelvin	K/10	K • 10 <sup>-1</sup>
1207 hex	decidegree Rankine	°R/10	K • 1.8 • 10 <sup>-1</sup>
1208-12FF hex	reserved for future standardization		

<sup>2</sup> The second is an SI base unit, defined as the duration of 9192631770 periods of the radiation corresponding to the transition between the two hyperfine levels of the ground state of the cesium-133 atom.

<sup>3</sup> When the *millisecond* unit is encoded with INT data type, it is equivalent to the ITIME data type defined in Appendix C. When the *millisecond* unit is encoded with DINT data type, it is equivalent to the TIME data type defined in Appendix C.

<sup>4</sup> When the *microsecond* unit is encoded with DINT data type, it is equivalent to the FTIME data type defined in Appendix C. When the *microsecond* unit is encoded with LINT data type, it is equivalent to the LTIME data type defined in Appendix C.

<sup>5</sup> The *DATE* unit and its symbol refer to the CIP DATE data type defined in Appendix C.

<sup>6</sup> The *TIME\_OF\_DAY* unit and its symbol refer to the CIP TIME\_OF\_DAY data type defined in Appendix C.

<sup>7</sup> The *DATE\_AND\_TIME* unit and its symbol refer to the CIP DATE\_AND\_TIME data type defined in Appendix C.

<sup>8</sup> Although the correct SI name is *degree Celcius*, this unit is often named *degree centigrade*.

<sup>9</sup> The kelvin is an SI base unit, defined as the fraction 1/273.16 of the thermodynamic temperature of the triple point of water.

## D-2.6 Pressure

Value	Name	Symbol	Base Units
1300 hex	pound-force per square inch (psi)	psi	$6.894757 \bullet 10^3 \bullet \text{Pa}$
1301 hex	torr	Torr	$(101325/760) \bullet \text{Pa}$
1302 hex	millitorr	mTorr	$(101325/760) \bullet 10^{-3} \bullet \text{Pa}$
1303 hex	millimeter of mercury (at 0°C)	mmHg (0°C)	
1304 hex	inch of mercury (at 0°C)	inHg (0°C)	$3.38638 \bullet 10^3 \bullet \text{Pa}$
1305 hex	centimeter of water (at 25°C)	cmH <sub>2</sub> O (25°C)	
1306 hex	inch of water (at 25°C)	inH <sub>2</sub> O (25°C)	
1307 hex	bar	bar	$10^5 \bullet \text{Pa}$
1308 hex	millibar	mbar	$10^2 \bullet \text{Pa}$
1309 hex	pascal	Pa	$\text{m}^{-1} \bullet \text{kg} \bullet \text{s}^{-2}$
130A hex	kilopascal	kPa	$10^3 \bullet \text{Pa}$
130B hex	standard atmosphere	atm	$101325 \bullet \text{Pa}$
130C hex	gram-force per square centimeter	gf/cm <sup>2</sup>	$9.80665 \bullet 10^1 \bullet \text{Pa}$
130D hex	millimeter of water (at 25°C)	mmH <sub>2</sub> O (25°C)	
130E-13FF hex	reserved for future standardization		

## D-2.7 Flow

Value	Name	Symbol	Base Units
1400 hex	standard cubic centimeter per minute	SCCM	
1401 hex	standard liter per minute	SLM	
1402 hex	cubic foot per minute	CFM	$\text{ft}^3/\text{min}$
1403 hex	pascal-cubic meter per second	(Pa • m <sup>3</sup> )/s	
1404 hex	kilogram per second	kg/s	
1405 hex	cubic meter per second	m <sup>3</sup> /s	
1406 hex	liter per second	L/s	$10^{-3} \bullet \text{m}^3/\text{s}$
1407 hex	milliliter per second	mL/s	$10^{-6} \bullet \text{m}^3/\text{s}$
1408 hex	gallon per second	GPS	gal/s
1409 hex	gallon per minute	GPM	gal/min
140A hex	gallon per hour	GPH	gal/hr
140B hex	pound per second		lb/s
140C hex	pound per minute		lb/min
140D hex	pound per hour		lb/h
140E hex	milligrams per minute	mg/M	$10^{-3} \bullet \text{g}/\text{min}$
140F hex	grams per minute	g/M	g/min
1410 hex	kilograms per hour	kg/H	$10^3 \bullet \text{g}/\text{hr}$
1411 hex	milliliter per minute	mL/m	$10^{-6} \bullet \text{m}^3/\text{min}$
1412 hex	milliliter per hour	mL/h	$10^{-6} \bullet \text{m}^3/\text{hr}$
1413 hex	liter per minute	L/m	$10^{-3} \bullet \text{m}^3/\text{min}$
1414 hex	liter per hour	L/h	$10^{-3} \bullet \text{m}^3/\text{hr}$
1415-14FF hex	reserved for future standardization		

### D-2.8 Acceleration

Value	Name	Symbol	Base Units
1500 hex	meter per second squared	$\text{m/s}^2$	
1501 hex	foot per second squared	$\text{ft/s}^2$	
1502 hex	inch per second squared	$\text{in/s}^2$	
1503 hex	angular acceleration	$\text{rad/s}^2$	
1504 hex	acceleration during free fall (gravity)	$\text{g}_n$	$9.80665 \bullet \text{m/s}^2$
1505-15FF hex	reserved for future standardization		

### D-2.9 Amount of Substance

Value	Name	Symbol	Base Units
1600 hex	mole <sup>#10</sup>	mol	
1601 hex	Refer to table D-2.34		
1602-16FF hex	reserved for future standardization		

### D-2.10 Angle

Value	Name	Symbol	Base-Units
1700 hex	radian (plane angle) <sup>#11</sup>	rad	
1701 hex	steradian (solid angle) <sup>#12</sup>	sr	
1702 hex	revolution	r	$6.283185 \bullet \text{rad}$
1703 hex	degree	$^\circ$	$(\pi/180) \bullet \text{rad}$
1704 hex	arc minute (minute)	'	$(1/60)^\circ$
1705 hex	arc second (second)	''	$(1/60)'$
1706-17FF hex	reserved for future standardization		

### D-2.11 Area

Value	Name	Symbol	Base Units
1800 hex	square meter	$\text{m}^2$	
1801 hex	square centimeter	$\text{cm}^2$	
1802 hex	square kilometer	$\text{km}^2$	
1803 hex	square yard	$\text{yd}^2$	
1804 hex	square foot	$\text{ft}^2$	
1805 hex	square inch	$\text{in}^2$	
1806 hex	square mile	$\text{mi}^2$	
1807-18FF hex	reserved for future standardization		

<sup>10</sup> The mole is an SI base unit, defined as the amount of substance of a system which contains as many elementary entities as there are atoms in 0.012 kilogram of carbon 12. When the mole is used, the elementary entities must be specified and may be atoms, molecules, ions, electrons, other particles, or specified groups of such particles.

<sup>11</sup> The radian is defined as the plane angle between two radii of a circle that cuts off on the circumference an arc equal in length to the radius.

<sup>12</sup> The steradian is defined as the solid angle that, having its vertex in the center of a sphere, cuts off an area of the surface of the sphere equal to that of a square with sides of length equal to the radius of the sphere.

### D-2.12 Capacitance

Value	Name	Symbol	Base Units
1900 hex	farad	F	$\text{m}^{-2} \bullet \text{kg}^{-1} \bullet \text{s}^4 \bullet \text{A}^2$
1901 hex	millifarad	mF	$10^{-3} \bullet \text{F}$
1902 hex	microfarad	$\mu\text{F}$	$10^{-6} \bullet \text{F}$
1903 hex	nano farad	nF	$10^{-9} \bullet \text{F}$
1904 hex	picofarad	pF	$10^{-12} \bullet \text{F}$
1905 hex	femtofarad	fF	$10^{-15} \bullet \text{F}$
1906 hex	permittivity	F/m	$\text{m}^{-3} \bullet \text{kg}^{-1} \bullet \text{s}^4 \bullet \text{A}^2$
1907-19FF hex	reserved for future standardization		

### D-2.13 Charge (Electric Charge)

Value	Name	Symbol	Base Units
1A00 hex	coulomb	C	A • s
1A01 hex	millicoulomb	mC	$10^{-3} \bullet \text{C}$
1A02 hex	microcoulomb	$\mu\text{C}$	$10^{-6} \bullet \text{C}$
1A03 hex	nanocoulomb	nC	$10^{-9} \bullet \text{C}$
1A04 hex	picocoulomb	pC	$10^{-12} \bullet \text{C}$
1A05 hex	femtocoulomb	fC	$10^{-15} \bullet \text{C}$
1A06 hex	ampere hour	A • h	A • s • 3600
1A07 hex	exposure (x and $\gamma$ rays)	C/kg	$\text{kg}^{-1} \bullet \text{A} \bullet \text{s}$
1A08-1AFF hex	reserved for future standardization		

### D-2.14 Conductance (Electric Conductance)

Value	Name	Symbol	Base Units
1B00 hex	siemens	S	$\text{m}^{-2} \bullet \text{kg}^{-1} \bullet \text{s}^3 \bullet \text{A}^2$
1B01 hex	millisiemens	mS	$10^{-3} \bullet \text{S}$
1B02 hex	microsiemens	$\mu\text{S}$	$10^{-6} \bullet \text{S}$
1B03 hex	nanosiemens	nS	$10^{-9} \bullet \text{S}$
1B04 hex	picosiemens	pS	$10^{-12} \bullet \text{S}$
1B05 hex	femtosiemens	fS	$10^{-15} \bullet \text{S}$
1B06-1BFF hex	reserved for future standardization		

### D-2.15 Current (Electric Current)

Value	Name	Symbol	Base Units
1C00 hex	ampere # <sup>13</sup>	A	
1C01 hex	100 milliamp, deciamp	100mA	$10^{-1} \bullet A$
1C02 hex	milliamp	mA	$10^{-3} \bullet A$
1C03 hex	microamp	$\mu A$	$10^{-6} \bullet A$
1C04 hex	nanoamp	nA	$10^{-9} \bullet A$
1C05 hex	picoamp	pA	$10^{-12} \bullet A$
1C06 hex	femtoamp	fA	$10^{-15} \bullet A$
1C07 hex	kiloamp	kA	$10^3 \bullet A$
1C08 hex	megaamp	MA	$10^6 \bullet A$
1C09 hex	gigaamp	GA	$10^9 \bullet A$
1C0A hex	magnetic field strength	A/m	
1C0B-1CFF hex	reserved for future standardization		

### D-2.16 Energy (Heat, Work)

Value	Name	Symbol	Base Units
1D00 hex	joule	J	$m^2 \bullet kg \bullet s^{-2}$
1D01 hex	watt second	W • s	J
1D02 hex	watt hour	W • h	$3.6 \bullet 10^3 \bullet J$
1D03 hex	kilowatt hour	kW • h	$3.6 \bullet 10^6 \bullet J$
1D04 hex	calorie (thermochemical, nutrition)	cal	4.184 • J
1D05 hex	British thermal unit	Btu	1005.056 • J
1D06 hex	kiloBtu	kBtu	$1005.056 \bullet 10^3 \bullet J$
1D07 hex	megaBtu	MBtu	$1005.056 \bullet 10^6 \bullet J$
1D08 hex	foot pound-force (foot-pound)	ft • lbf	1.355818 • J
1D09 hex	electronvolt # <sup>14</sup>	eV	$1.60217733 \bullet 10^{-19} \bullet J$
1D0A hex	heat capacity (entropy)	J/K	$m^2 \bullet kg \bullet s^{-2} \bullet K^{-1}$
1D0B hex	British thermal unit per degree Fahrenheit	Btu/°F	$1.899101 \bullet 10^3 \bullet J/K$
1D0C hex	specific heat capacity (specific entropy)	J/(kg • K)	$m^2 \bullet s^{-2} \bullet K^{-1}$
1D0D hex	specific energy	J/kg	$m^2 \bullet s^{-2}$
1D0E hex	energy density	J/m <sup>3</sup>	$m^{-1} \bullet kg \bullet s^{-2}$
1D0F hex	molar energy	J/mol	$m^2 \bullet kg \bullet s^{-2} \bullet mol^{-1}$
1D10 hex	molar entropy (molar heat capacity)	J/(mol • K)	$m^2 \bullet kg \bullet s^{-2} \bullet K^{-1} \bullet mol^{-1}$
1D11-1DFF hex	reserved for future standardization		

<sup>13</sup> The ampere is an SI base unit, defined as the constant current which, if maintained in two straight parallel conductors of infinite length, of negligible circular cross section, and placed 1 meter apart in vacuum, would produce between these conductors a force equal to  $2 \times 10^{-7}$  newton per meter of length.

<sup>14</sup> The electronvolt is a unit which is accepted for use with SI units. However, the accepted value for 1eV has been experimentally obtained and the base units shown does not represent an exact value.

### D-2.17 Force

Value	Name	Symbol	Base Units
1E00 hex	newton	N	$m \bullet kg \bullet s^{-2}$
1E01 hex	surface tension	N/m	$kg \bullet s^{-2}$
1E02 hex	kilogram-force	kgf	$9.80665 \bullet N$
1E03 hex	pound-force	lbf	$4.448 \bullet N$
1E04 hex	ounce-force	ozf	$0.278 \bullet N$
1E05-1EFF hex	reserved for future standardization		

### D-2.18 Frequency (includes RPM)<sup>#15</sup>

Value	Name	Symbol	Base Units
1F00 hex	hertz	Hz	$s^{-1}$
1F01 hex	kilohertz	kHz	$10^3 \bullet s^{-1}$
1F02 hex	megahertz	MHz	$10^6 \bullet s^{-1}$
1F03 hex	gigahertz	GHz	$10^9 \bullet s^{-1}$
1F04 hex	counts per second	CPS	$s^{-1}$
1F05 hex	counts per millisecond		$10^3 \bullet s^{-1}$
1F06 hex	counts per microsecond		$10^6 \bullet s^{-1}$
1F07 hex	counts per minute	CPM	$s^{-1} / 60$
1F08 hex	counts per hour		$s^{-1} / 3600$
1F09 hex	counts per day		$s^{-1} / 86400$
1F0A hex	baud rate	baud <sup>#16</sup>	bit/s
1F0B hex	kbaud	kbaud	kbit/s
1F0C hex	Mbaud	Mbaud	Mbit/s
1F0D hex	angular velocity	rad/s	
1F0E hex	revolution per second (rps)	rps <sup>#17</sup>	
1F0F hex	revolution per minute (rpm)	rpm <sup>#18</sup>	$1.047198 \bullet 10^{-1} \bullet rad/s$
1F10 hex	revolution per hour	r/h	
1F11 hex	revolution per day	r/d	
1F12-1FFF hex	reserved for future standardization		

<sup>15</sup> For the purposes of this appendix, *Frequency* is defined as rate of occurrence (for a specified event).

<sup>16</sup> Usage of the symbols *Bd* and *bd* is also acceptable.

<sup>17</sup> Usage of the symbols *RPS* and *r/s* is also acceptable.

<sup>18</sup> Usage of the symbols *RPM* and *r/min* is also acceptable.

**D-2.19 Inductance**

Value	Name	Symbol	Base Units
2000 hex	henry	H	$\text{m}^2 \bullet \text{kg} \bullet \text{s}^{-2} \bullet \text{A}^{-2}$
2001 hex	millihenry	mH	$10^{-3} \bullet \text{H}$
2002 hex	microhenry	$\mu\text{H}$	$10^{-6} \bullet \text{H}$
2003 hex	nano henry	nH	$10^{-9} \bullet \text{H}$
2004 hex	picohenry	pH	$10^{-12} \bullet \text{H}$
2005 hex	femtohenry	fH	$10^{-15} \bullet \text{H}$
2006 hex	permeability	H/m	$\text{m} \bullet \text{kg} \bullet \text{s}^{-2} \bullet \text{A}^{-2}$
2007-20FF hex	reserved for future standardization		

**D-2.20 Inertia**

Value	Name	Symbol	Base Units
2100 hex	inertia (as $\text{kg} \bullet \text{m}^2$ )	$\text{kg} \bullet \text{m}^2$	
2101 hex	inertia (as $\text{mg} \bullet \text{m}^2$ )	$\text{mg} \bullet \text{m}^2$	$10^{-6} \bullet \text{kg} \bullet \text{m}^2$
2102-21FF hex	reserved for future standardization		

**D-2.21 Length (Distance, Displacement)**

Value	Name	Symbol	Base Units
2200 hex	meter <sup>#19</sup>	m	
2201 hex	kilometer	km	$10^3 \bullet \text{m}$
2202 hex	centimeter	cm	$10^{-2} \bullet \text{m}$
2203 hex	millimeter	mm	$10^{-3} \bullet \text{m}$
2204 hex	micron (micrometer)	$\mu$	$10^{-6} \bullet \text{m}$
2205 hex	nanometer	nm	$10^{-9} \bullet \text{m}$
2206 hex	angstrom	A	$10^{-10} \bullet \text{m}$
2207 hex	inch	in	$2.54 \bullet 10^{-2} \bullet \text{m}$
2208 hex	foot	ft	$3.048 \bullet 10^{-1} \bullet \text{m}$
2209 hex	yard	yd	$9.144 \bullet 10^{-1} \bullet \text{m}$
220A hex	mile	mi	$1.609344 \bullet 10^3 \bullet \text{m}$
220B hex	nautical mile	nm <sup>#20</sup>	$1.852 \bullet 10^3 \bullet \text{m}$
220C hex	astronomical unit	AU	$1.495979 \bullet 10^{11} \bullet \text{m}$
220D hex	light year	l.y.	$9.46073 \bullet 10^{15} \bullet \text{m}$
220E hex	parsec	pc	$3.085678 \bullet 10^{16} \bullet \text{m}$
220F hex	point (computer)		(1/72) in
2210 hex	point (printer's)		$3.514598 \bullet 10^{-4} \bullet \text{m}$
2211-22FF hex	reserved for future standardization		

<sup>19</sup> The meter is an SI base unit, defined as the length of the path traveled by light in vacuum during a time interval of 1/299792458 second.

<sup>20</sup> When using the symbol for nautical mile (nm), care should be taken to avoid confusion with the symbol for nanometer.

## D-2.22 Light (Luminous Intensity)

Value	Name	Symbol	Base Units
2300 hex	candela <sup>#21</sup>	cd	
2301 hex	luminance	cd/m <sup>2</sup>	
2302 hex	luminous flux (lumen)	lm	cd • sr
2303 hex	illuminance (lux, lumen/m <sup>2</sup> )	lx	m <sup>-2</sup> • cd • sr
2304 hex	footcandle		10.76391 • lx
2305-23FF hex	reserved for future standardization		

## D-2.23 Magnetic Flux

Value	Name	Symbol	Base Units
2400 hex	weber	Wb	m <sup>2</sup> • kg • s <sup>-2</sup> • A <sup>-1</sup>
2401 hex	magnetic flux density (tesla)	T	kg • s <sup>-2</sup> • A <sup>-1</sup>
2402 hex	magnetic field (oersted)	Oe	79.57747 • A/m
2403-24FF hex	reserved for future standardization		

## D-2.24 Mass

Value	Name	Symbol	Base Units
2500 hex	kilogram <sup>#22</sup>	kg	
2501 hex	gram	g	10 <sup>-3</sup> • kg
2502 hex	milligram	mg	10 <sup>-6</sup> • kg
2503 hex	metric ton (megagram)	t	10 <sup>3</sup> • kg
2504 hex	ounce (avoirdupois)	oz	2.834952 • 10 <sup>-2</sup> • kg
2505 hex	pound (avoirdupois)	lb	4.535924 • 10 <sup>-1</sup> • kg
2506 hex	short ton (2000 lb)		9.071847 • 10 <sup>2</sup> • kg
2507-25FF hex	reserved for future standardization		

<sup>21</sup> The candela is an SI base unit, defined as the luminous intensity, in a given direction, of a source that emits monochromatic radiation of frequency  $540 \times 10^{12}$  hertz and that has a radiant intensity in that direction of (1/683) watt per steradian.

<sup>22</sup> The kilogram is an SI base unit, defined as the mass of the international prototype of the kilogram.

### D-2.25 Power (Wattage)

Value	Name	Symbol	Base Units
2600 hex	watt	W	$\text{m}^2 \bullet \text{kg} \bullet \text{s}^{-3}$
2601 hex	milliwatt	mW	$10^{-3} \bullet \text{W}$
2602 hex	microwatt	$\mu\text{W}$	$10^{-6} \bullet \text{W}$
2603 hex	nanowatt	nW	$10^{-9} \bullet \text{W}$
2604 hex	picowatt	pW	$10^{-12} \bullet \text{W}$
2605 hex	femtowatt	fW	$10^{-15} \bullet \text{W}$
2606 hex	kilowatt	kW	$10^3 \bullet \text{W}$
2607 hex	British thermal unit per hour	Btu/h	$2.930711 \bullet 10^{-1} \bullet \text{W}$
2608 hex	erg per second	erg/s	$10^{-7} \bullet \text{W}$
2609 hex	horsepower (electric, automotive)		$7.46 \bullet 10^2 \bullet \text{W}$
260A hex	horsepower (boiler)		$9.8095 \bullet 10^3 \bullet \text{W}$
260B hex	British thermal unit per square foot hour	Btu/(ft <sup>2</sup> • h)	$3.154591 \bullet \text{W/m}^2$
260C hex	radiant intensity	W/sr	$\text{m}^2 \bullet \text{kg} \bullet \text{s}^{-3} \bullet \text{sr}^{-1}$
260D hex	radiance	W/(m <sup>2</sup> • sr)	$\text{kg} \bullet \text{s}^{-3} \bullet \text{sr}^{-1}$
260E hex	thermal conductivity	W/(m • K)	$\text{m} \bullet \text{kg} \bullet \text{s}^{-3} \bullet \text{K}^{-1}$
260F-26FF hex	reserved for future standardization		

### D-2.26 Radiology

Value	Name	Symbol	Base Units
2700 hex	activity of a radionuclide (becquerel)	Bq	s <sup>-1</sup>
2701 hex	absorbed dose (gray)	Gy	$\text{m}^2 \bullet \text{s}^{-2}$
2702 hex	dose equivalent (sievert)	Sv	$\text{m}^2 \bullet \text{s}^{-2}$
2703 hex	absorbed dose rate	Gy/s	$\text{m}^2 \bullet \text{s}^{-3}$
2704 hex	curie	Ci	$3.7 \bullet 10^{10} \bullet \text{Bq}$
2705 hex	roentgen	R	$2.58 \bullet 10^{-4} \bullet \text{C/kg}$
2706 hex	rad	rad <sup>#23</sup>	$10^{-2} \bullet \text{Gy}$
2707 hex	rad equivalent man	rem	$10^{-2} \bullet \text{Sv}$
2708-27FF hex	reserved for future standardization		

<sup>23</sup> When there is risk of confusion with the symbol for the radian, rd may be used as the symbol for rad.

**D-2.27 Resistance (Electric Resistance)**

Value	Name	Symbol	Base Units
2800 hex	ohm	$\Omega$	$m^2 \bullet kg \bullet s^{-3} \bullet A^{-2}$
2801 hex	milliohm	$m\Omega$	$10^{-3} \bullet \Omega$
2802 hex	microohm	$\mu\Omega$	$10^{-6} \bullet \Omega$
2803 hex	nanoohm	$n\Omega$	$10^{-9} \bullet \Omega$
2804 hex	picoohm	$p\Omega$	$10^{-12} \bullet \Omega$
2805 hex	femtoohm	$f\Omega$	$10^{-15} \bullet \Omega$
2806 hex	kilohm	$k\Omega$	$10^3 \bullet \Omega$
2807 hex	megaohm	$M\Omega$	$10^6 \bullet \Omega$
2808 hex	gigaohm	$G\Omega$	$10^9 \bullet \Omega$
2809 hex	ohm centimeter	$\Omega \bullet cm$	$10^{-2} \bullet \Omega \bullet m$
280A-28FF hex	reserved for future standardization		

**D-2.28 Sound**

Value	Name	Symbol	Base Units
2900 hex	bel	B	
2901 hex	decibel	dB	$10^{-1} \bullet B$
2902 hex	Intensity ( $W \bullet m^{-2}$ )	I	
2903 hex	Sound Intensity Level	IL	see #24
2904 hex	Sound Pressure Level	SPL	see #25
2905 hex	Voltage Level re 1mW across 600 $\Omega$	dB(mW)	see #26
2906 hex	Voltage Level re 1 V	dB(V)	see #27
2907 hex	Voltage Level re 1 $\mu$ V	dB( $\mu$ V)	
2908-29FF hex	reserved for future standardization		

**D-2.29 Torque (Moment of Force)**

Value	Name	Symbol	Base Units
2A00 hex	torque (moment of force)	N $\bullet$ m	$m^2 \bullet kg \bullet s^{-2}$
2A01 hex	torque (mili)	$10^{-3} \bullet N \bullet m$	
2A02 hex	dyne centimeter	dyn $\bullet$ cm	$10^{-7} \bullet N \bullet m$
2A03 hex	kilogram-force meter	kgf $\bullet$ m	9.80665 $\bullet$ N $\bullet$ m
2A04 hex	ounce-force inch (ounce-inch)	ozf $\bullet$ in	$7.0615 \bullet 10^{-3} \bullet N \bullet m$
2A05 hex	pound-force inch (pound-inch)	lbf $\bullet$ in	$1.1298 \bullet 10^{-1} \bullet N \bullet m$
2A06 hex	pound-force foot (pound-foot)	lbf $\bullet$ ft	
2A07 hex	torque per ampere	(N $\bullet$ m)/A	
2A08 hex	torque per ampere (mili)	$10^{-3} \bullet ((N \bullet m)/A)$	
2A09-2AFF hex	reserved for future standardization		

<sup>24</sup> Sound Intensity Level is  $10 \bullet log \bullet (I/I_o)$ , where  $I_o = 10^{-12} W/m^2$  in air and  $6.66 \cdot 10^{-19} W/m^2$  in water.

<sup>25</sup> Sound Pressure Level is  $20 \bullet log \bullet (P/P_o)$ , where  $P_o = 20 \mu Pa$  in air and  $1 \mu Pa$  in water.

<sup>26</sup> Voltage Level re 1mW across 600 $\Omega$  is  $20 \bullet log \bullet (V/.775)$ , when the reference impedance is 600 $\Omega$ .

<sup>27</sup> Voltage Level re 1 V is  $20 \bullet log \bullet (V/1)$ , where the impedance ratio is generally ignored.

**D-2.30 Velocity (Speed)<sup>#28</sup>**

Value	Name	Symbol	Base Units
2B00 hex	meter per second	m/s	
2B01 hex	centimeter per second	cm/s	
2B02 hex	kilometer per hour	km/h	$2.7777 \bullet 10^{-1} \bullet \text{m/s}$
2B03 hex	speed of light	c	$3 \bullet 10^8 \bullet \text{m/s}$
2B04 hex	mile per hour	mi/h	$4.4704 \bullet 10^{-1} \bullet \text{m/s}$
2B05 hex	knot (nautical mile per hour)	kt	$(1852/3600) \bullet \text{m/s}$
2B06 hex	foot per second	ft/s	
2B07 hex	inch per second	in/s	
2B08-2BFF hex	reserved for future standardization		

**D-2.31 Viscosity**

Value	Name	Symbol	Base Units
2C00 hex	dynamic viscosity	Pa • s	$\text{m}^{-1} \bullet \text{kg} \bullet \text{s}^{-1}$
2C01 hex	poise	P	$10^{-1} \bullet \text{Pa} \bullet \text{s}$
2C02 hex	centipoise	cP	$10^{-3} \bullet \text{Pa} \bullet \text{s}$
2C03-2CFF hex	reserved for future standardization		

**D-2.32 Voltage**

Value	Name	Symbol	Base Units
2D00 hex	volt	V	$\text{m}^2 \bullet \text{kg} \bullet \text{s}^{-3} \bullet \text{A}^{-1}$
2D01 hex	millivolt	mV	$10^{-3} \bullet \text{V}$
2D02 hex	microvolt	µV	$10^{-6} \bullet \text{V}$
2D03 hex	nanovolt	nV	$10^{-9} \bullet \text{V}$
2D04 hex	picovolt	pV	$10^{-12} \bullet \text{V}$
2D05 hex	femtovolt	fV	$10^{-15} \bullet \text{V}$
2D06 hex	kilovolt	kV	$10^3 \bullet \text{V}$
2D07 hex	megavolt	MV	$10^6 \bullet \text{V}$
2D08 hex	gigavolt	GV	$10^9 \bullet \text{V}$
2D09 hex	electric field strength	V/m	$\text{m} \bullet \text{kg} \bullet \text{s}^{-3} \bullet \text{A}^{-1}$
2D0A-2DFF hex	reserved for future standardization		

<sup>28</sup> For the purposes of this appendix, *Velocity* uses the traditional definition of distance over time, and thus units for *baud rate* and *rpm* are listed in the *Frequency* group.

**D-2.33      Volume**

Value	Name	Symbol	Base Units
2E01 hex	cubic meter	$\text{m}^3$	
2E02 hex	liter	L <sup>29</sup>	$10^{-3} \bullet \text{m}^3$
2E03 hex	milliliter	$\text{mL}$	$10^{-6} \bullet \text{m}^3$
2E04 hex	kiloliter	$\text{kL}$	$\text{m}^3$
2E05 hex	cubic yard	$\text{yd}^3$	$7.645549 \bullet 10^{-1} \bullet \text{m}^3$
2E06 hex	cubic foot	$\text{ft}^3$	$2.831685 \bullet 10^{-2} \bullet \text{m}^3$
2E07 hex	cubic inch	$\text{in}^3$	$1.638706 \bullet 10^{-5} \bullet \text{m}^3$
2E08 hex	gallon (U.S.)	gal	$\text{L} \bullet 3.785412 \bullet \text{L}$
2E09 hex	quart (U.S. liquid)	liq qt	$9.463529 \bullet 10^{-1} \bullet \text{L}$
2E0A hex	pint (U.S. liquid)	pt	$4.731765 \bullet 10^{-1} \bullet \text{L}$
2E0B hex	ounce (U.S. fluid)	fl oz	$2.957353 \bullet 10^1 \bullet \text{mL}$
2E0C hex	barrel (U.S.)	bbl	42 gal
2E0D hex	specific volume	$\text{m}^3/\text{kg}$	
2E0E-2EFF hex	reserved for future standardization		

**D-2.34      Density**

Value	Name	Symbol	Base Units
1601 hex	mole per cubic meter	$\text{mol}/\text{m}^3$	
2F01 hex	watt per square meter	$\text{W}/\text{m}^2$	$\text{kg} \bullet \text{s}^{-3}$
2F02 hex	joule per cubic meter	$\text{J}/\text{m}^3$	$\text{m}^{-1} \bullet \text{kg} \bullet \text{s}^{-2}$
2F03 hex	electric charge density	$\text{C}/\text{m}^3$	$\text{m}^{-3} \bullet \text{s} \bullet \text{A}$
2F04 hex	electric flux density	$\text{C}/\text{m}^2$	$\text{m}^{-2} \bullet \text{s} \bullet \text{A}$
2F05 hex	current density	$\text{A}/\text{m}^2$	
2F06 hex	British thermal unit per cubic foot	$\text{Btu}/\text{ft}^3$	$3.725895 \bullet 10^4 \bullet \text{J}/\text{m}^3$
2F07 hex	mass density	$\text{kg}/\text{m}^3$	
2F08 hex	mass density	$\text{g}/\text{cm}^3$	$10^3 \bullet \text{kg}/\text{m}^3$
2F09 hex	counts per milliliter	$1/\text{mL}$	
2F0A hex	counts per gallon	$1/\text{gal}$	
2F0B hex	pounds per gallon	$\text{lb}/\text{gal}$	
2F0C hex	pounds per cubic foot	$\text{lb}/\text{ft}^3$	
2F0D hex	pounds per cubic inch	$\text{lb}/\text{in}^3$	
2F0E hex	grams per milliliter	$\text{g}/\text{mL}$	$10^6 \bullet \text{g}/\text{m}^3$
2F0F hex	grams per liter	$\text{g}/\text{L}$	$10^3 \bullet \text{g}/\text{m}^3$
2F10 hex	kilograms per liter	$\text{kg}/\text{L}$	$10^6 \bullet \text{g}/\text{m}^3$
2F11 hex	micrograms per liter	$\mu\text{g}/\text{L}$	$10^{-3} \bullet \text{g}/\text{m}^3$
2F12 hex	micrograms per cubic meter	$\mu\text{g}/\text{m}^3$	$10^{-6} \bullet \text{g}/\text{m}^3$
2F13-2FFF hex	reserved for future standardization		

<sup>29</sup> The letter L was adopted by the General Conference on Weights and Measures (CGPM) in order to avoid the risk of confusion between the letter l and the numeral 1. Both the letter l and the letter L are internationally accepted symbols for the liter. According to [Interpretation of the SI for the United States and Metric Conversion Policy for Federal Agencies](#) (National Institute for Standards and Technology Pub 814), to avoid this risk, the symbol to be used in the United States is the letter L.