

# **CIP Common Specification**

---

Volume 1

Release 1.0

June 5, 2001

ControlNet International

and

Open DeviceNet Vendor Association

This page is intentionally left blank

# CIP Common Specification

## Volume 1

### Table of Contents

<b>Chapter 1</b>	- Introduction to the Control and Information Protocol
<b>Chapter 2</b>	- Messaging Protocol
<b>Chapter 3</b>	- Communications Objects
<b>Chapter 4</b>	- How to Read Specifications in the Object Library
<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

## **Volume 1: CIP Common Specification**

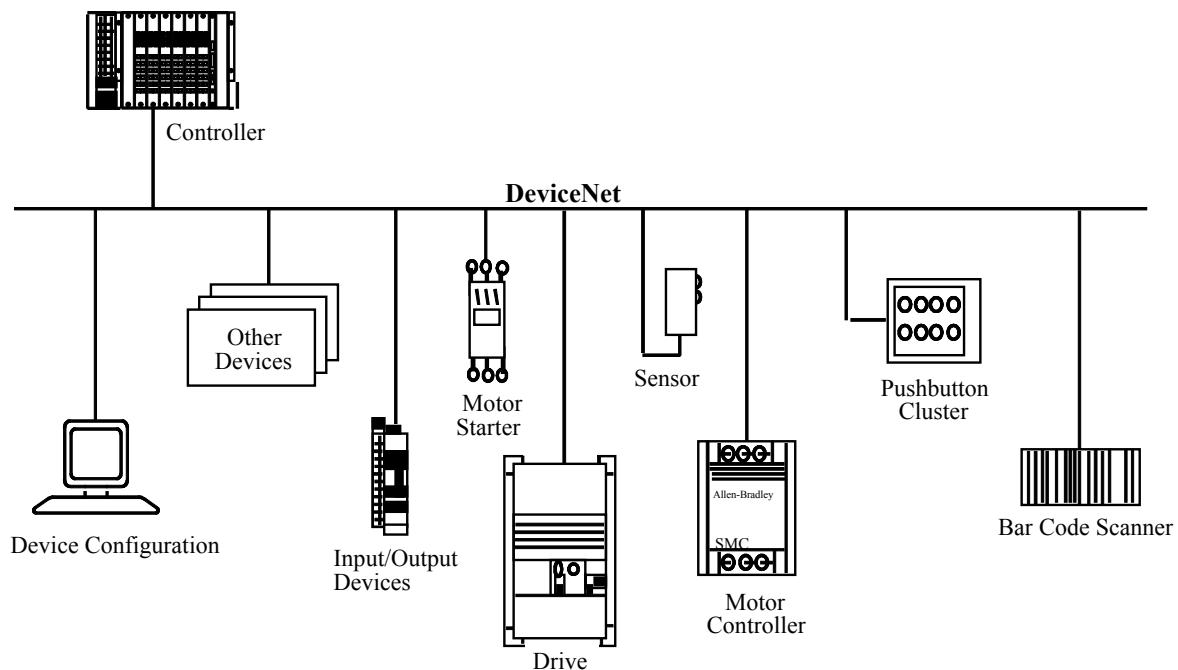
### **Chapter 1: Introduction to CIP**

This page is intentionally left blank

## 1-1 INTRODUCTION

The Control and Information 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 which is related to the system being controlled, such as configuration parameters and diagnostics.

## 1-2 OBJECT MODELING

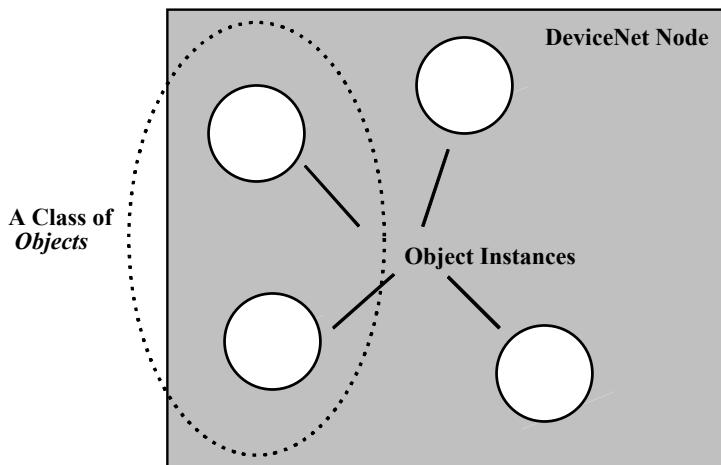
CIP makes use of abstract *object modeling* to describe:

- 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 DeviceNet 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.

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

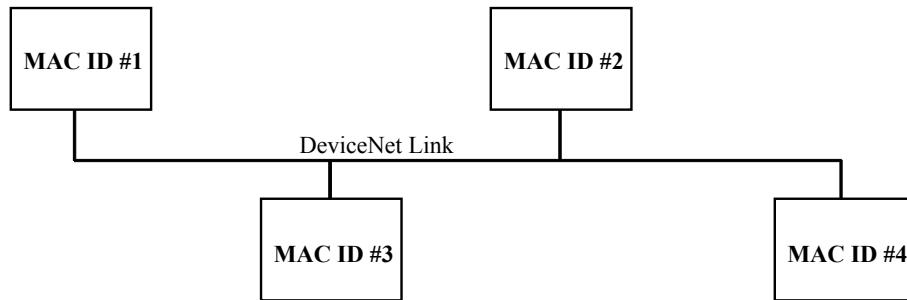
The following Object Modeling related terms are used when describing DeviceNet services and protocol.

- **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.

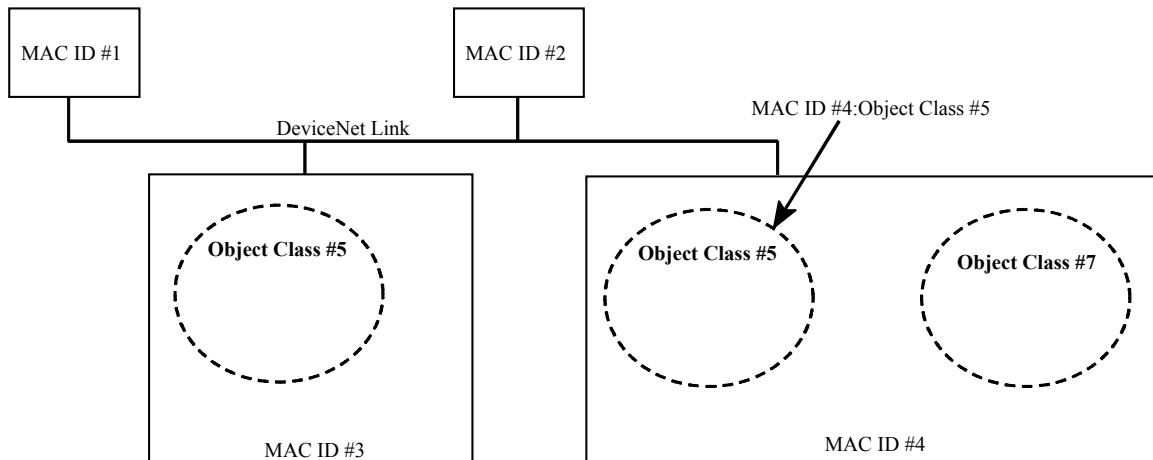
## 1-2.1 Object Addressing

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:

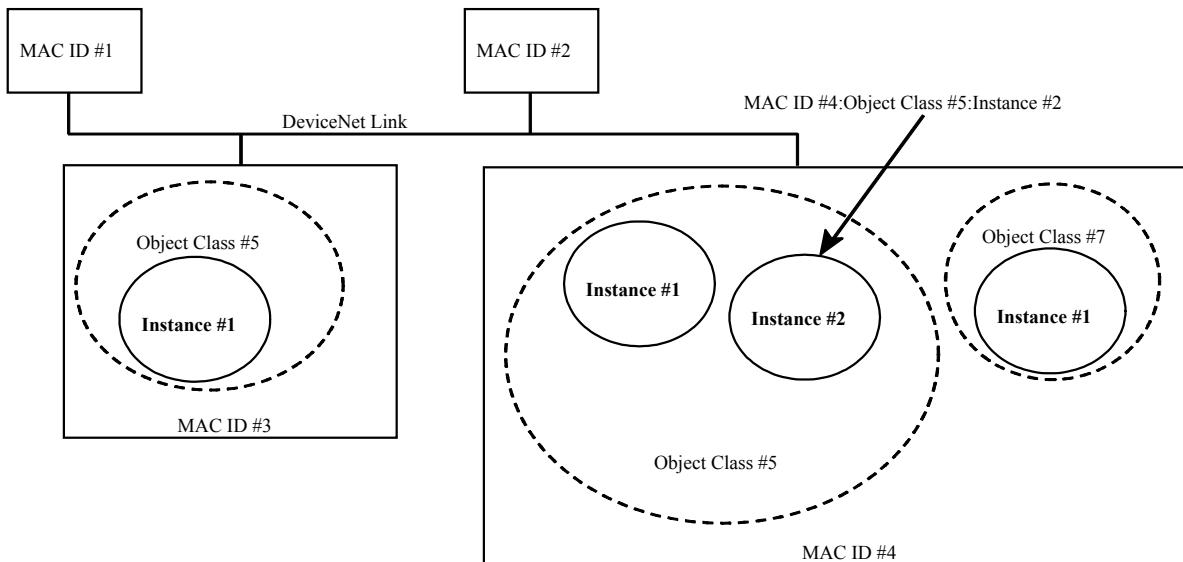
- **Media Access Control Identifier (MAC ID)** - An integer identification value assigned to each node on the CIP network. This value distinguishes a node among all other nodes on the same link.



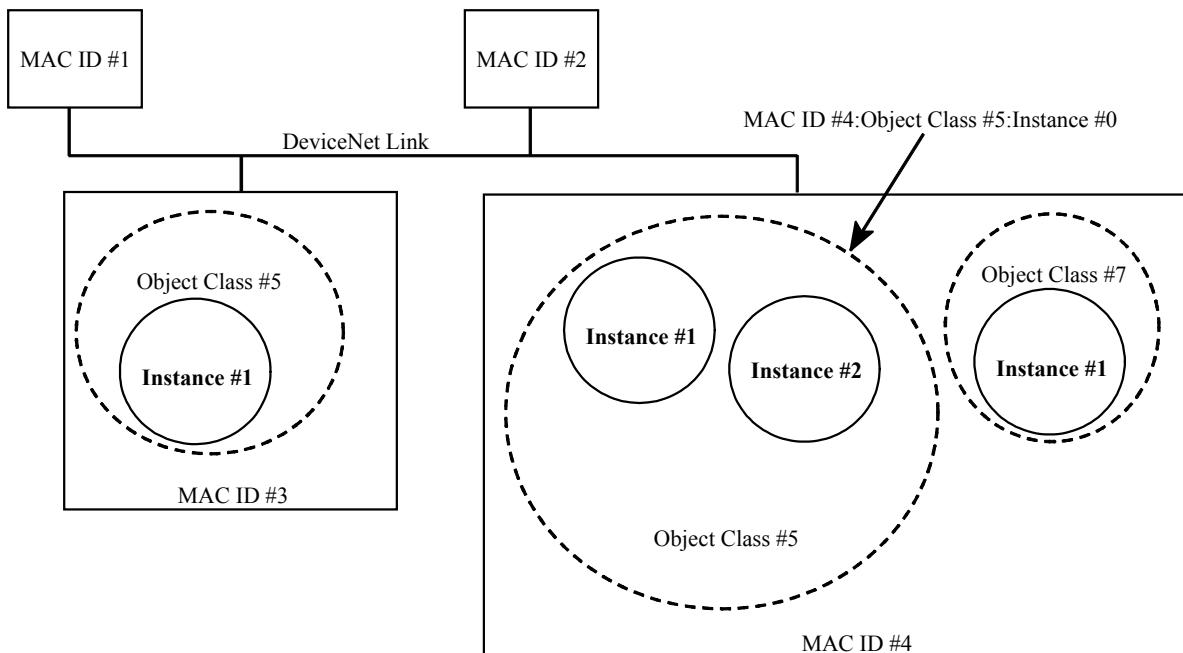
- **Class Identifier (Class ID)** - An integer identification value assigned to each *Object Class* accessible from the network.



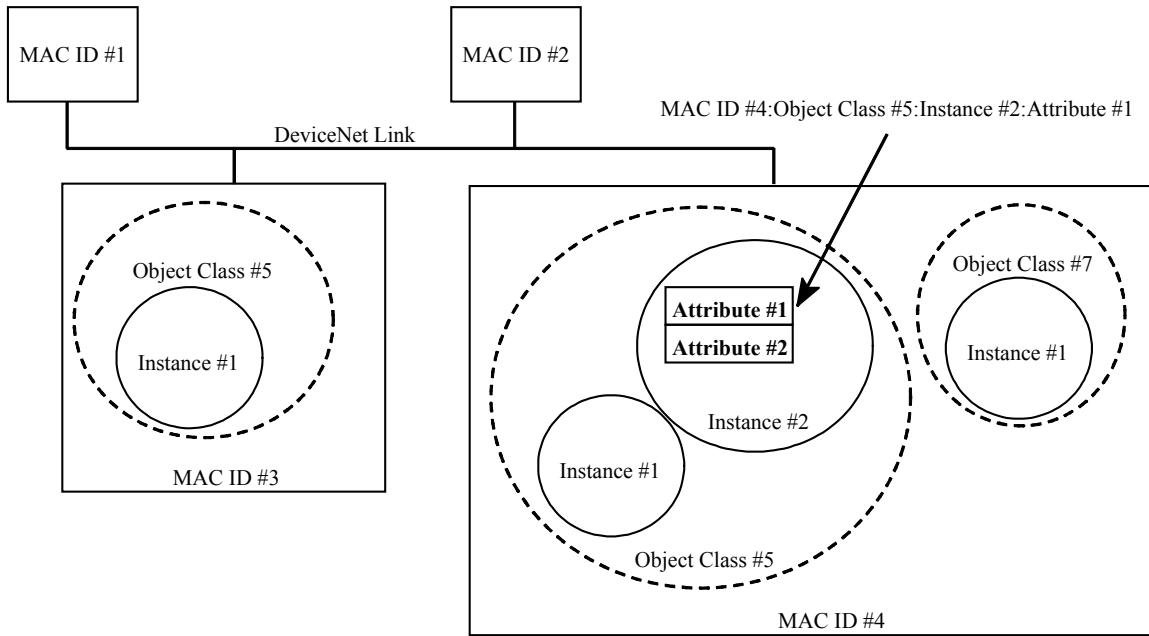
- **Instance Identifier (Instance ID)** - An integer identification value assigned to an *Object Instance* that identifies it among all *Instances* of the same *Class*. This integer is unique within the *MAC ID:Class* in which it resides.



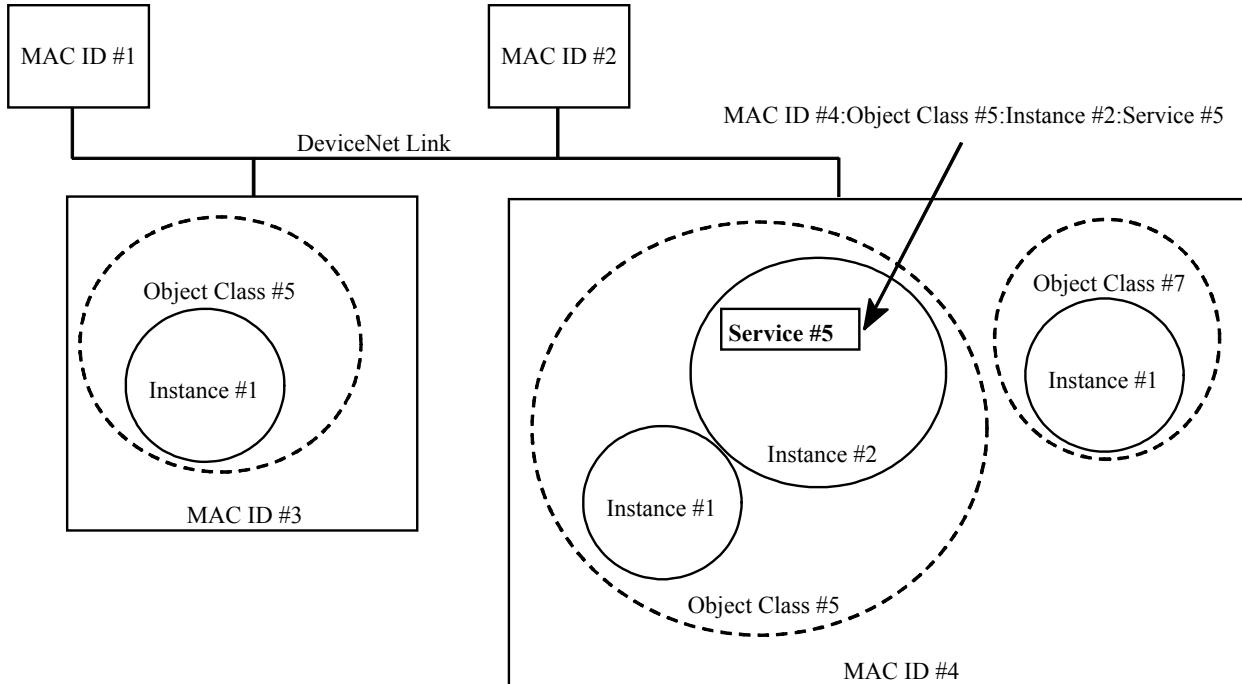
It is also possible to address the Class itself versus a specific Object Instance within the Class. This is accomplished by utilizing the Instance ID value zero (0). **CIP reserves the Instance ID value zero (0) to indicate a reference to the Class versus a specific Instance within the Class.**



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



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



## 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> - FF <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 ?.
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

## 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:

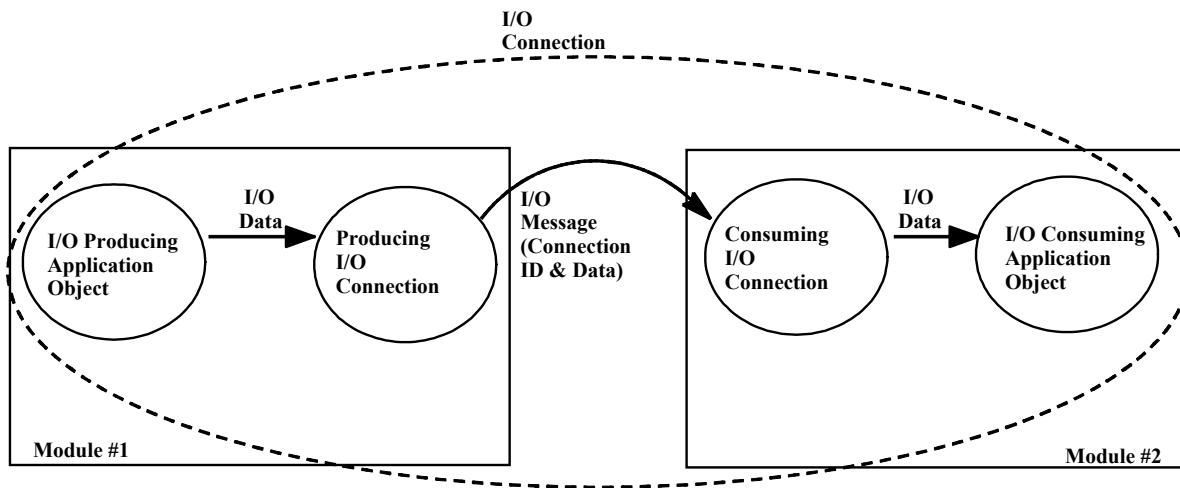
- **I/O Connections** - Provide dedicated, special-purpose communication paths between a producing application and one or more consuming applications. Application-specific I/O data moves through these ports and is often referred to implicit messaging.
- **Explicit Messaging Connections** - Provide generic, 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 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.

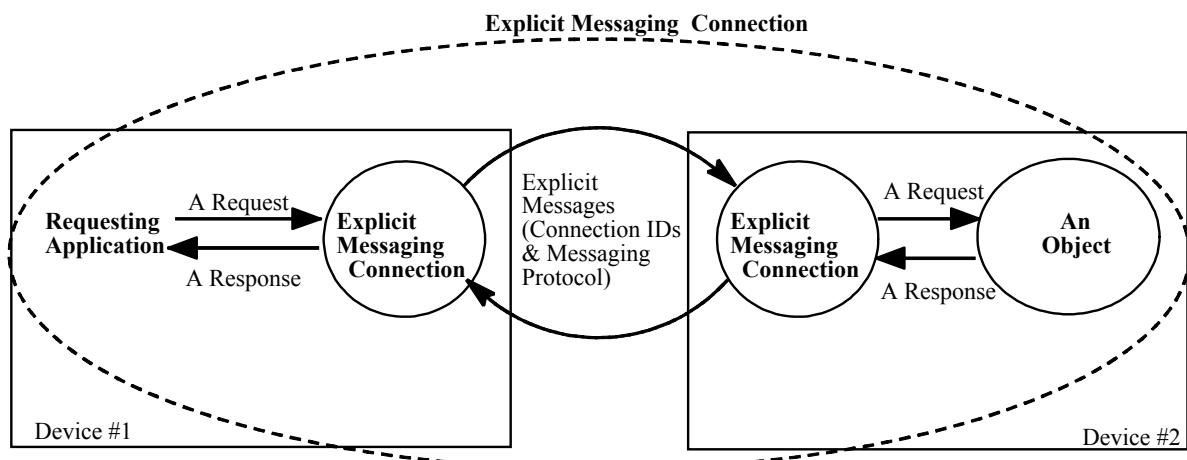
### 1-3.2 Explicit Messaging Connections

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.

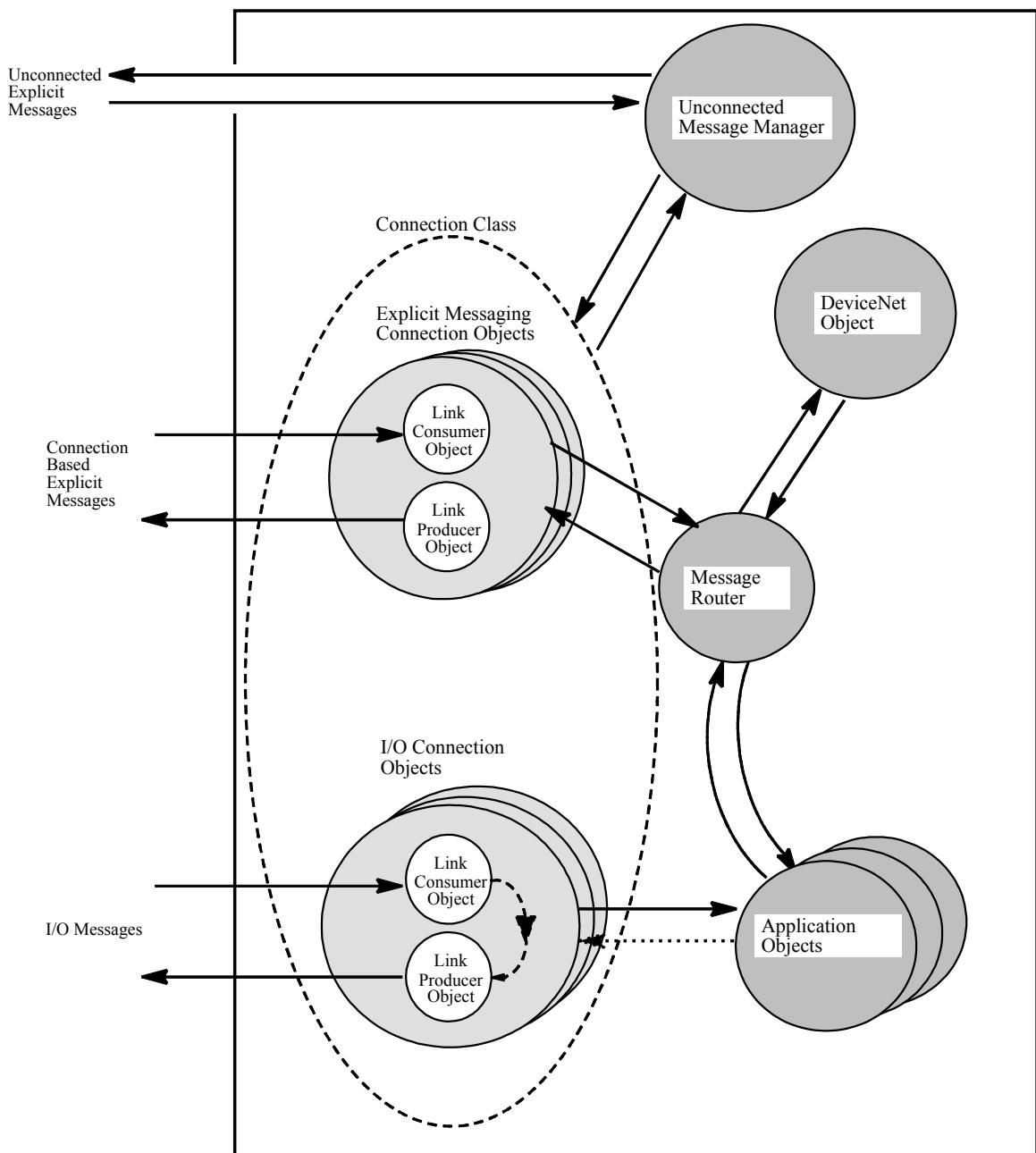
**Figure 1-3.2. CIP Explicit Messaging Connection**



## 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 Class**- Allocates and manages internal resources associated with both I/O and Explicit Messaging connections.
- **Connection Object** - Manages the communication-specific aspects associated with a particular application-to-application network relationship.
- **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.

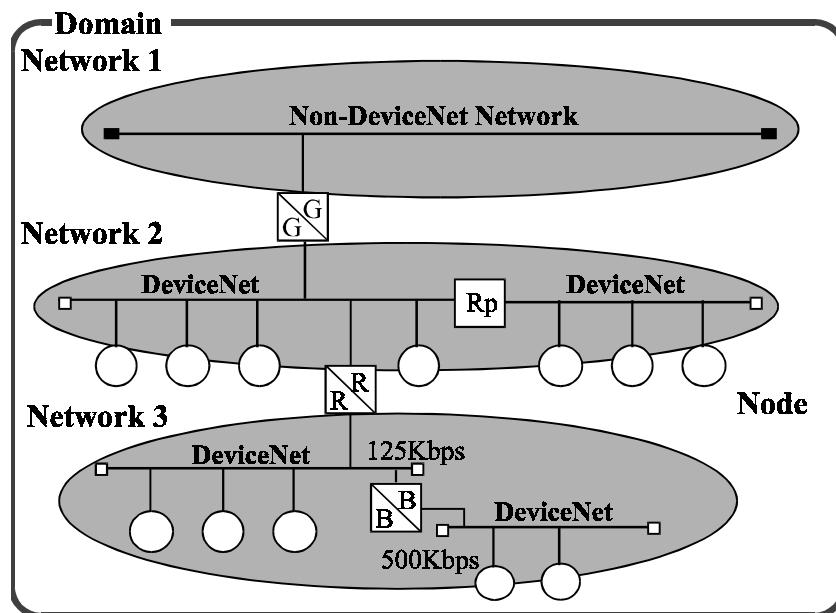
**Figure 1-4.1.** CIP Module Object Model

## 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 which 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

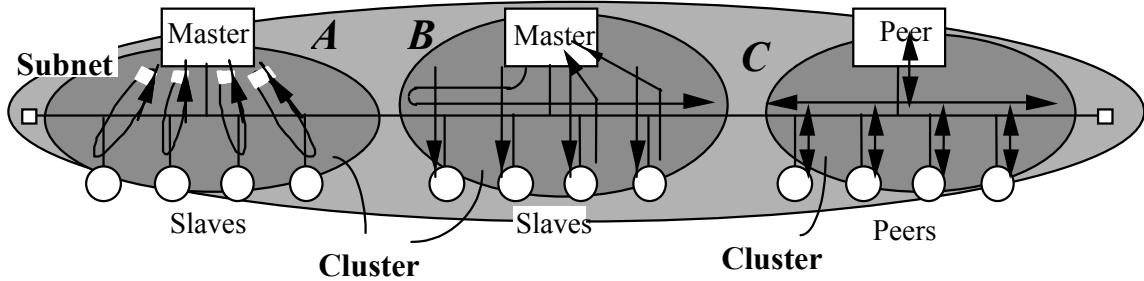
- **Rp** **Segment Repeater** between segments
  - Segments participate in the same media arbitration
- **B** **Subnet Bridge** between subnets
  - Duplicate MAC ID check passes through
  - MAC ID's on one subnet may not be duplicated on the other subnet
  - Subnets may operate at different baud rates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

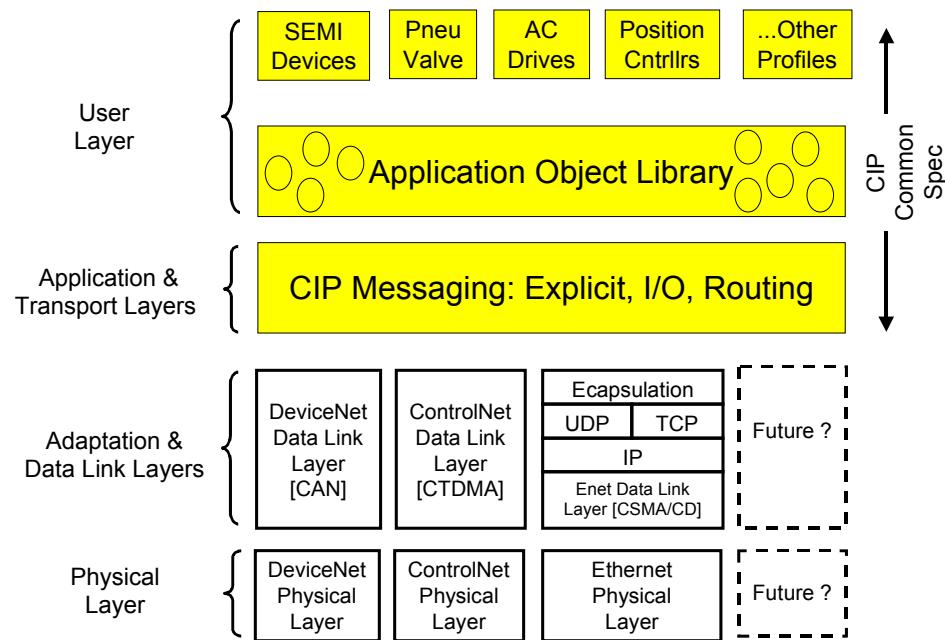
**Cluster C** - Peer-to-peer Communication (point-to-point or multicast)

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

## 1-6 CIP SPECIFICATION STRUCTURE

This specification (CIP Common) is the definition of the application and user layers for a number of CIP networks. The figure below shows the relationship between the specifications of this document and those of the CIP networks.

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



## 1-7 DEFINITIONS

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

<b>1-7.1 actual packet interval (API)</b>	The measure of how frequently a specific connection produces its data.
<b>1-7.2 allocate</b>	To take a resource from a common area and assign that resource for the exclusive use of a specific entity.
<b>1-7.3 application</b>	Function or data structure for which data is consumed or produced.
<b>1-7.4 application objects</b>	Multiple object classes that manage and provide the run-time exchange of messages across the network and within the network device.
<b>1-7.5 attribute</b>	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 behaviour of an object. Attributes are divided into class attributes and instance attributes.
<b>1-7.6 behaviour</b>	Indication of how the object responds to particular events. Its description includes the relationship between attribute values and services.
<b>1-7.7 big endian</b>	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]
<b>1-7.8 bit</b>	A unit of information consisting of a 1 or a 0. This is the smallest data unit that can be transmitted.
<b>1-7.9 byte</b>	See octet.
<b>1-7.10 class</b>	A set of objects all of which represent a similar system component. A class is a generalisation of the object, a template for defining variables and methods. All objects in a class are identical in form and behaviour, but they may contain different attribute values.
<b>1-7.11 class specific service</b>	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.
<b>1-7.12 client</b>	(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.
<b>1-7.13 communication objects</b>	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.
<b>1-7.14 connection</b>	A logical binding between two application objects. These application objects may be the same or different devices.
<b>1-7.15 connection ID (CID)</b>	Identifier assigned to a transmission that is associated with a particular connection between producers and consumers that identifies a specific piece of application information.
<b>1-7.16 connection path</b>	The attribute is made up of a byte stream that defines the application object to which a connection instance applies.
<b>1-7.17 consume</b>	The act of receiving data from a producer.
<b>1-7.18 consumer</b>	A node that is receiving data from a producer.
<b>1-7.19 consuming application</b>	The application that consumes data.
<b>1-7.20 cyclic</b>	Term used to describe events that repeat in a regular and repetitive manner.
<b>1-7.21 device</b>	A physical hardware connection to the link. A device may contain more than one node.
<b>1-7.22 device profile</b>	A collection of device-dependent information and functionality providing consistency between similar devices of the same device type.
<b>1-7.23 end node</b>	A producing or consuming node.

<b>1-7.24 end point</b>	One of the communicating entities involved in a connection.
<b>1-7.25 error</b>	A discrepancy between a computed, observed or measured value or condition and the specified or theoretically correct value or condition.
<b>1-7.26 instance</b>	The actual physical presentation of an object within a class. Identifies one of many objects within the same object class.
<b>1-7.27 instantiated</b>	An object that has been created in a device.
<b>1-7.28 library element</b>	A derived or standard data type, function, function block, program or resource in IEC 1131 - programmable controllers.
<b>1-7.29 link</b>	Collection of nodes with unique MAC IDs. Segments connected by repeaters make up a link; links connected by routers make up a network.
<b>1-7.30 little endian</b>	Describes a model of memory organisation 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.
<b>1-7.31 Message Router</b>	The object within a node that distributes messaging requests to the appropriate application objects.
<b>1-7.32 multicast</b>	A packet with a special destination address, which multiple nodes on the network may be willing to receive. [Source: RFC1392]
<b>1-7.33 multicast connection</b>	A connection from one node to many. Multicast connections allow a single producer to be received by many consumer nodes.
<b>1-7.34 network</b>	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.
<b>1-7.35 network address or node address</b>	A node's 32-bit TCP/IP address on the link. In most CIP networks, this network address is the MAC ID; however, this is not the case on Ethernet. The DLL of Ethernet has a 48-bit MAC ID that is not used directly by the CIP communication stack.
<b>1-7.36 node</b>	A connection to a link that requires a single MAC ID.
<b>1-7.37 object</b>	(1) An abstract representation of a computer's capabilities. Objects can be composed of any or all of the following components: a) data (information which changes with time); b) configuration (parameters for behaviour); 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 behaviour.
<b>1-7.38 object specific service</b>	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.
<b>1-7.39 octet</b>	An octet is 8 bits that indicates no particular data type.
<b>1-7.40 originator</b>	The client responsible for establishing a connection path to the target.
<b>1-7.41 point-to-point connection</b>	A connection that exists between two nodes only. Connections can be either point-to-point or multicast.
<b>1-7.42 port</b>	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.
<b>1-7.43 produce</b>	Act of sending data to a consumer.
<b>1-7.44 producer</b>	A node that is responsible for transmitting data.
<b>1-7.45 redundant media</b>	A system using more than one medium to help prevent communication failures.
<b>1-7.46 requested packet interval (RPI)</b>	The measure of how frequently the originating application requires the transmission of data from the target application.
<b>1-7.47 serial number</b>	A unique 32-bit integer assigned by each manufacturer to every device. The number need only be unique with respect to the manufacturer.
<b>1-7.48 server</b>	An object that provides services to another (client) object.

<b>1-7.49 service</b>	Operation or function that an object performs upon request from another object.
<b>1-7.50 target</b>	The end-node to which a connection is established.
<b>1-7.51 tool</b>	An executable software program that interacts with the user to perform some function.
<b>1-7.52 unconnected message manager (UCMM)</b>	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>1-8.1 API</b>	actual packet interval
<b>1-8.2 ASCII</b>	American Standard Code for Information Interchange
<b>1-8.3 CIP</b>	The control and information protocol defined by the CIP Common Specification. CIP includes both connected and unconnected messaging.
<b>1-8.4 DLL</b>	Data Link Layer
<b>1-8.5 MAC ID</b>	the 48-bit physical address of an Ethernet node
<b>1-8.6 PDU</b>	protocol data unit
<b>1-8.7 O⇒T</b>	originator to target (used to describe packets that are sent from the originator to the target)
<b>1-8.8 OSI</b>	open systems interconnection (see ISO 7498)
<b>1-8.9 RPI</b>	requested packet interval
<b>1-8.10 SDU</b>	service data unit
<b>1-8.11 SEM</b>	state event matrix
<b>1-8.12 STD</b>	state transition diagram, used to describe object behaviour
<b>1-8.13 T⇒O</b>	target to originator (used to describe packets that are sent from the target to the originator)

This page is intentionally left blank

## **Volume 1: CIP Common Specification**

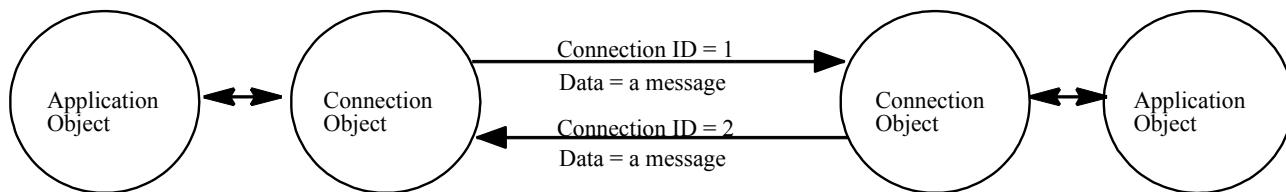
## **Chapter 2: Messaging Protocol**

This page is intentionally left blank

## 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

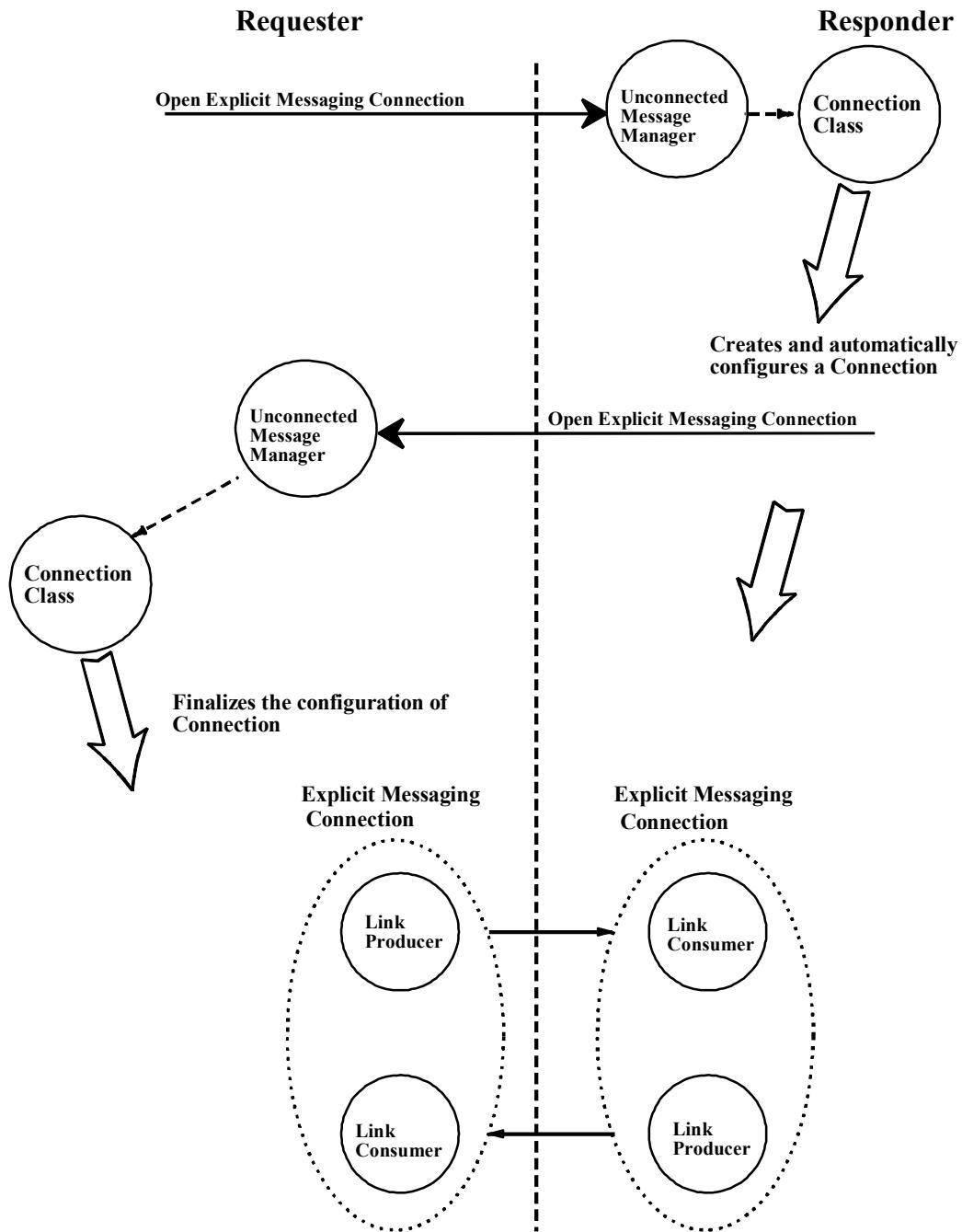
This section presents an overview of dynamically establishing both Explicit Messaging and I/O Connections.

### 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.

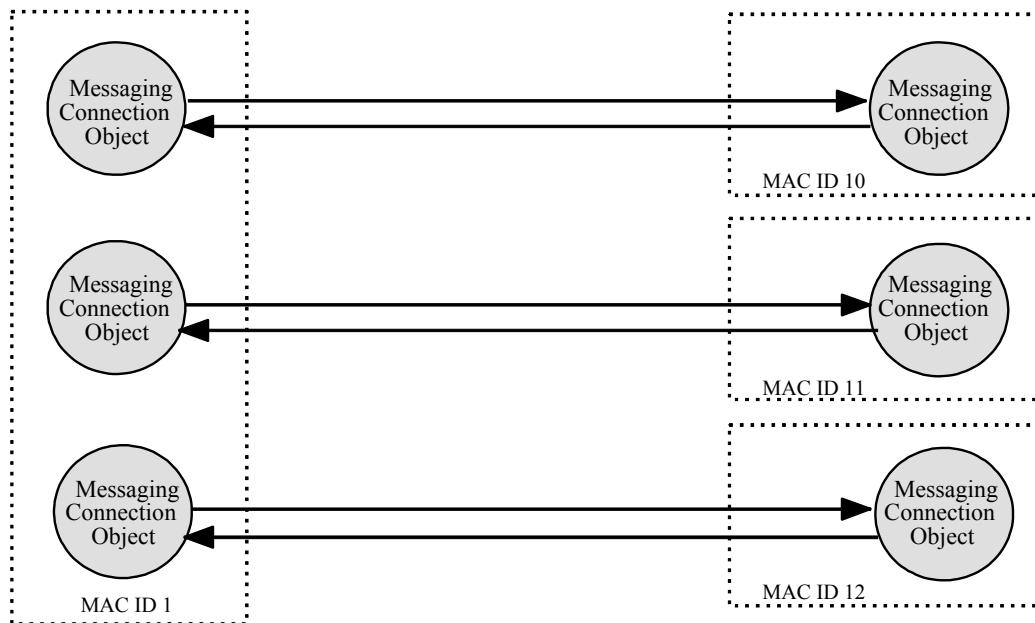
When using the UCMM to establish an explicit messaging connection, the target application object is the Message Router object (Class Code 2).

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

**Figure 2-2.1.** Establishing an Explicit Messaging Connection

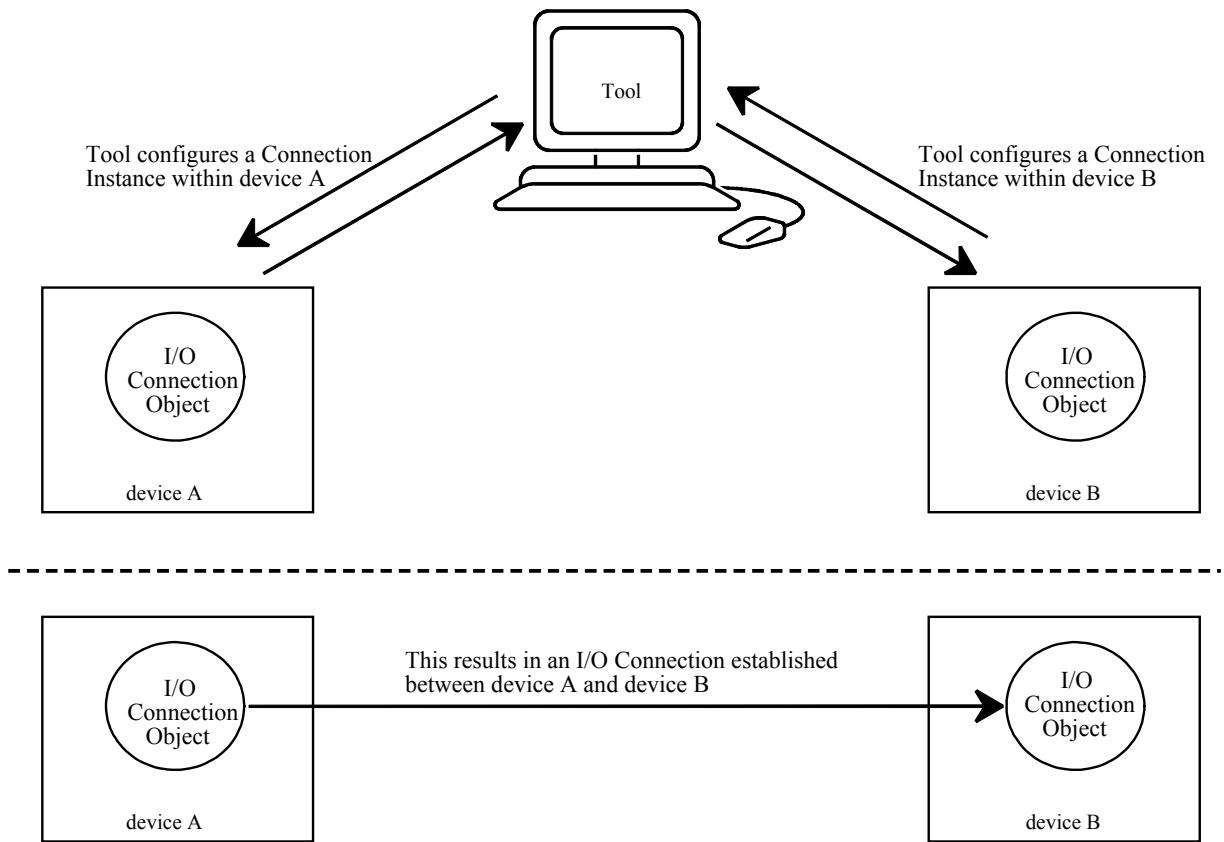
Explicit Messaging Connections are unconditionally ***point-to-point***. Point-to-point connections exist between two devices ONLY. The device that requests that the connection be opened (the client) is one *end-point* of the connection, and the module that receives and responds to the request (the server)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

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



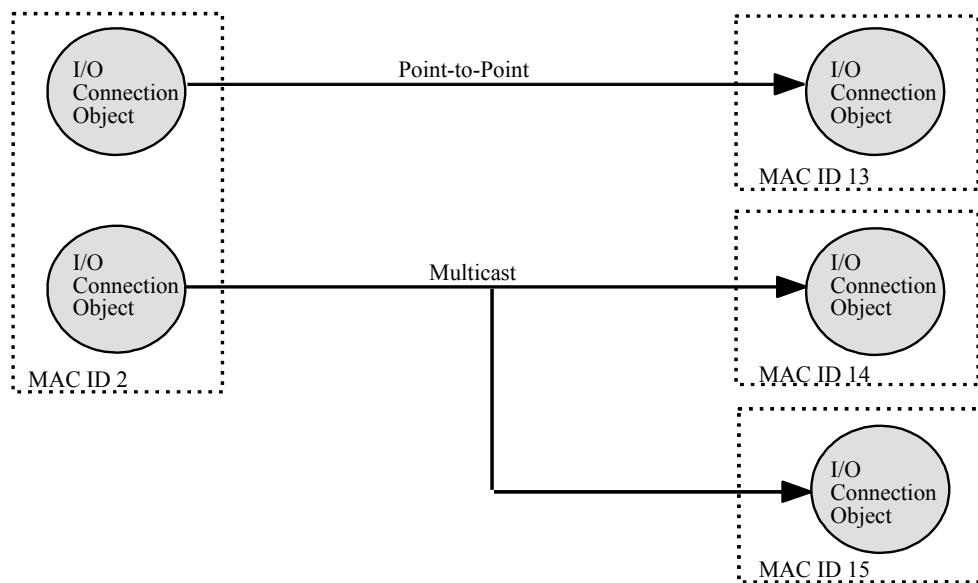
## 2-2.2 I/O Connections

The dynamic process facilitates 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.

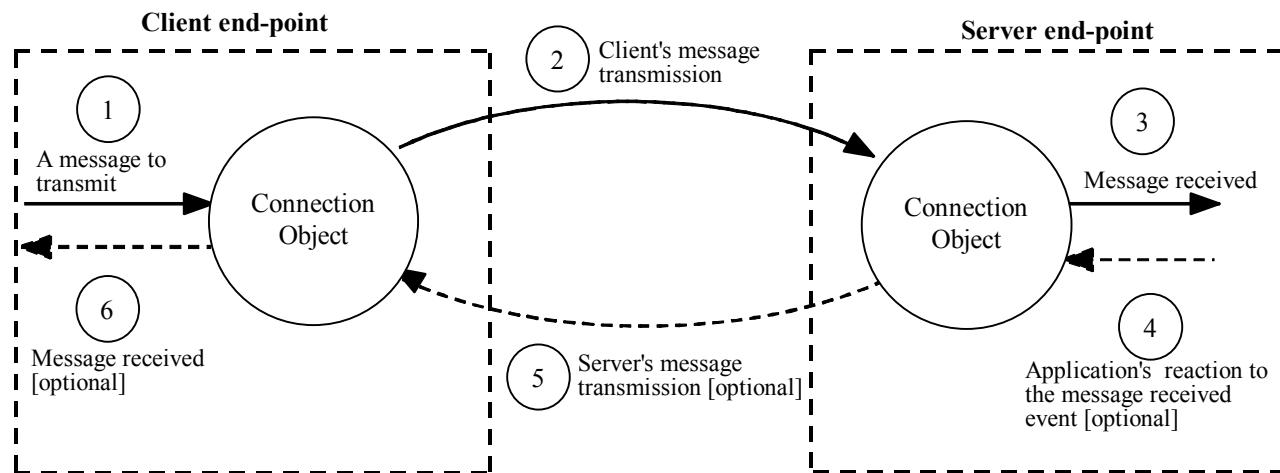
**Figure 2-2.3.** Tool Interface with Devices to Create Connection

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 END-POINTS**

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.



**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 as:

<b>Parameter Name</b>	<b>Data Type</b>	<b>Description</b>
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 as:

<b>Parameter Name</b>	<b>Data Type</b>	<b>Description</b>
Reply Service	UINT	Reply service code.
Reserved	USINT	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 UINT	Additional status.
Response Data	Array of octet	Response data from request or additional error data if General Status indicated an error.

## **Volume 1: CIP Common Specification**

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

This page is intentionally left blank

## 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

### 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.

- **USINT state** - 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.

- **UINT connection\_id** - The value placed within the CAN Identifier Field when this Link Producer is triggered to send. The Connection Object using this Link Producer internally initializes this attribute with the value in its **produced\_connection\_id** attribute. See section 3-4.3, Connection Instance Attributes for a definition of the produced\_connection\_id attribute.

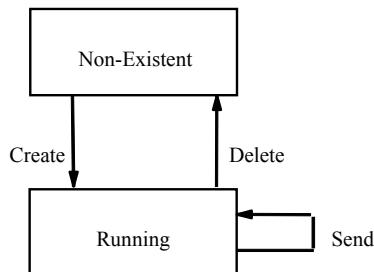
### 3-2.4 Link Producer Object Instance Services

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 3-2.2 and Table 3-2.3 illustrate the Link Producer's Instance behavior.

**Figure 3-2.2. Link Producer in Object State Transition****Table 3-2.3. 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 deletes a Link Consumer Object

#### 3-3.3 Link Consumer Instance Attributes

The following list describes the Link Consumer Instance attributes.

- **USINT state** - 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.

- **UINT connection\_id** - This attribute holds the *CAN Identifier* field value that denotes the message to be received by this consumer. The Connection Object utilizing this Link Consumer internally initializes this attribute with the value in its **consumed\_connection\_id** attribute. See section 3-4.3, Connection Instance Attributes for a definition of the consumed\_connection\_id attribute.

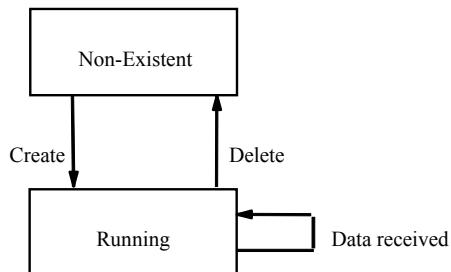
#### 3-3.4 Link Consumer Instance Services

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 3-3.2 and Table 3-3.3 illustrate the Link Consumer's Instance behavior.

**Figure 3-3.2. Link Consumer Object State Transition Diagram****Table 3-3.3. 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: 5**

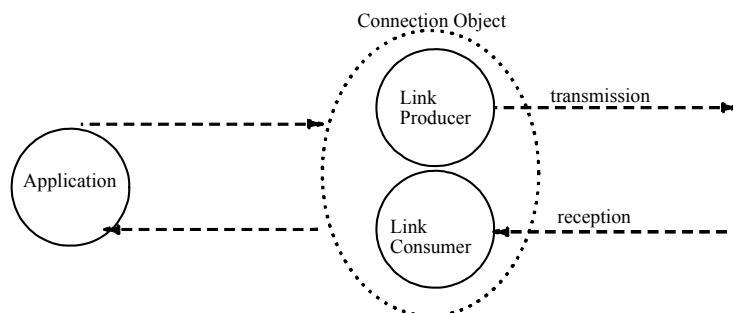
The Connection Class allocates and manages the internal resources associated with both I/O and Explicit Messaging Connections. The specific instance generated by the Connection Class is referred to as a *Connection Instance* or a *Connection Object*.

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 3-4.2.

**Table 3-4.2. 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.					

### 3-4.2 Connection Object Class Services

The Connection Class supports the following CIP Common Services:

**Table 3-4.3. Connection Class Services**

Service Code	Need In Implementation	Service Name	Service Description
08 <sub>hex</sub>	Optional	Create	Used to instantiate a Connection Object
09 <sub>hex</sub>	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.
05 <sub>hex</sub>	Optional	Reset	Used to reset all <i>resettable</i> Connection Objects. The conditions under which a Connection is <i>resettable</i> are listed in the State Event Matrix in section 3-4.6
11 <sub>hex</sub>	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.
0E <sub>hex</sub>	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 Object Instance Attributes

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

**Table 3-4.4. 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
5	Conditional	DeviceNet_consumed_connection_id	UINT	CAN Identifier Field value that denotes message to be received on a DeviceNet subnet.
6	Required	DeviceNet_initial_comm_characteristics	BYTE	Defines the Message Group(s) across which productions and consumptions associated with this Connection occur on a DeviceNet subnet.
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

Attr ID	Need In Implementation	Attribute Name	Data Type	Brief Description of Attribute
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.
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.

**Important:** Access Rules for the Connection Object Instance Attributes are defined in section 3-4.7.

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.

### 3-4.3.1 state Attribute - USINT data type

This attribute defines the current state of the Connection instance. Table 3-4.5 defines the possible states and assigns a value used to indicate that state. Also see figure 3-4.27 and figure 3-4.31 for state transition behavior.

**Table 3-4.5. 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 <b>may</b> be made to this state. See the <b>watchdog_timeout_action</b> attribute description and the description of the Inactivity/Watchdog Timer (section 3-4.4) for more details.
05	Deferred Delete <sup>2</sup>	If an Explicit Messaging Connection Object experiences an Inactivity/Watchdog timeout, then a transition <b>may</b> be made to this state. See the <b>watchdog_timeout_action</b> attribute description and the description of the Inactivity/Watchdog Timer (section 3-4.4) 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.

Value	State Name	Description
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.

**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*) breaks down 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 3-4.4, Connection Timing)

### 3-4.3.2 instance\_type Attribute - USINT data type

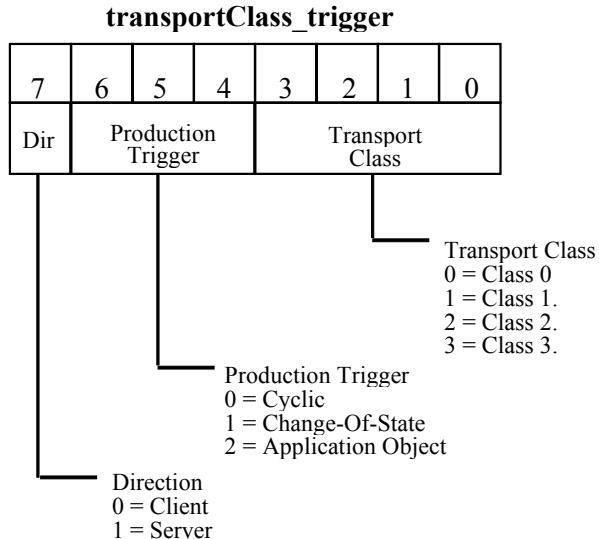
This attribute defines the instance type. See Table 3-4.6.

**Table 3-4.6. Values assigned to the instance\_type attribute**

Value	Meaning
00	<b>Explicit Messaging.</b> 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	<b>I/O.</b> 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	<b>CIP Bridged.</b> 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.

### 3-4.3.3. **transportClass\_trigger Attribute - USINT 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:



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 3-4.7. 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 <u>when</u> 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 3-4.8. Possible Values within Production Trigger Bits**

If the value is:	Then the Production of a message is:	
0	Cyclic	The expiration of the Transmission Trigger Timer triggers the data production. See section 3-4.4, 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 <b>expected_packet_rate</b> attribute of a Connection Object and the description of Connection Timing in section 3-4.4 for more information.

If the value is:	Then the Production of a message is:	
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 <b>expected_packet_rate</b> attribute of a Connection Object and the description of Connection Timing in section 3-4.4 for more information.
3 - 7	Reserved by CIP	

Table 3-4.9 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 3-4.9. Possible Values within Transport Class Bits**

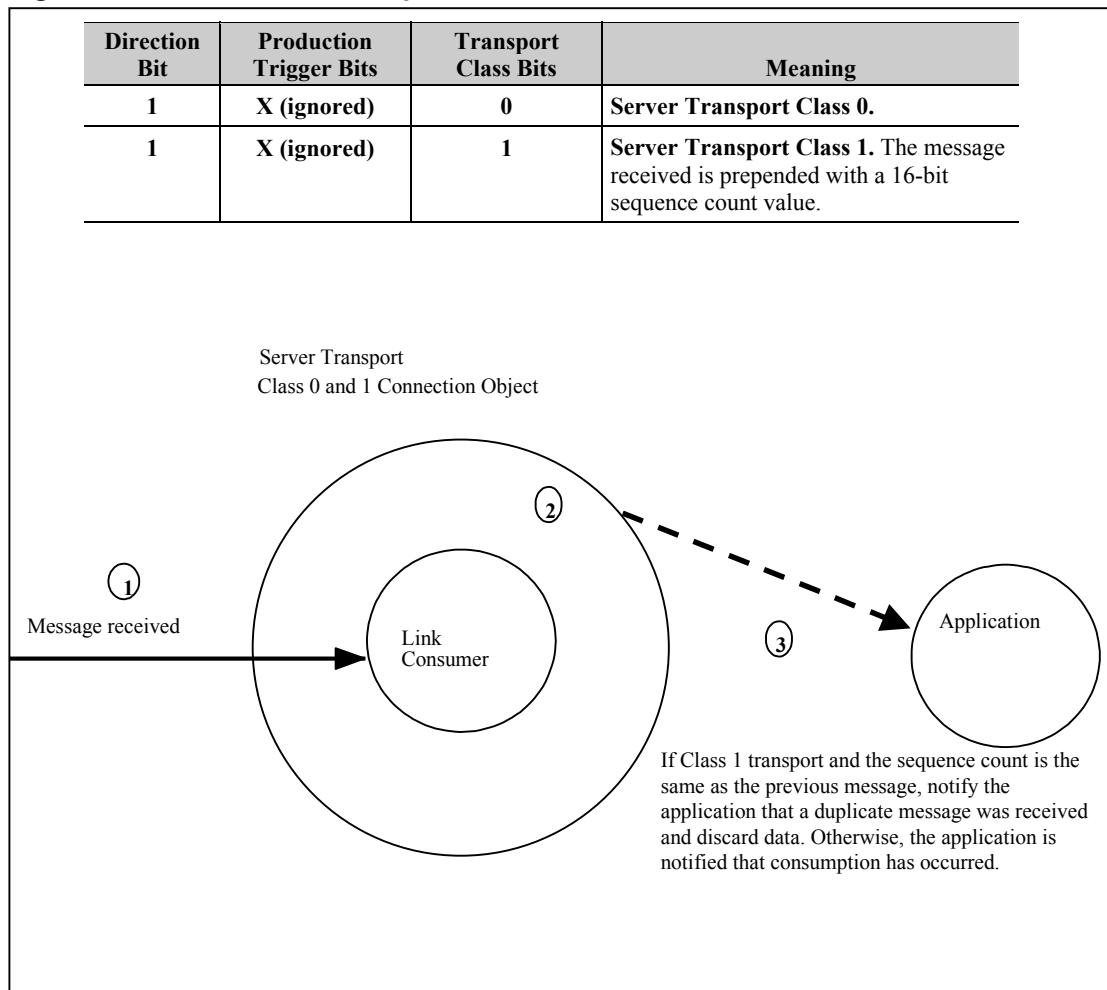
Value	Meaning	
0	Transport Class 0	Based on the value within the <i>Dir</i> bit, this connection end-point will be a producing only OR consuming only end-point. Upon application of this Connection instance, the module instantiates either a Link Producer ( <i>Dir</i> bit = Client, producing only) or a Link Consumer ( <i>Dir</i> 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 3-4.4 for a description of the **Transmission Trigger Timer**, which is shown in the illustrations.

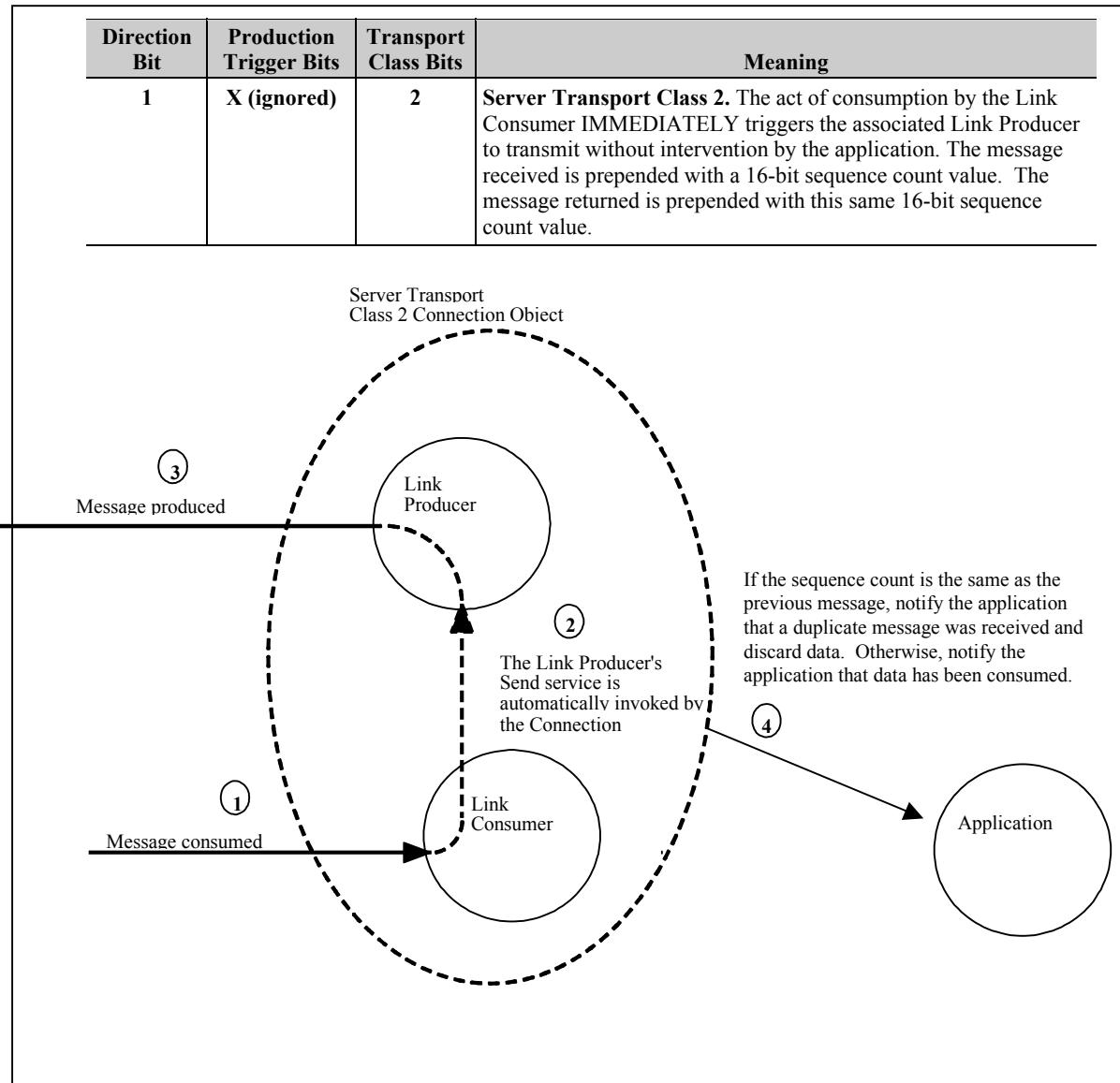
### 3-4.3.3.1 Server Transport Class 0 and 1 Behavior

**Figure 3-4.10. Server Transport Class 0 and 1 Behavior**



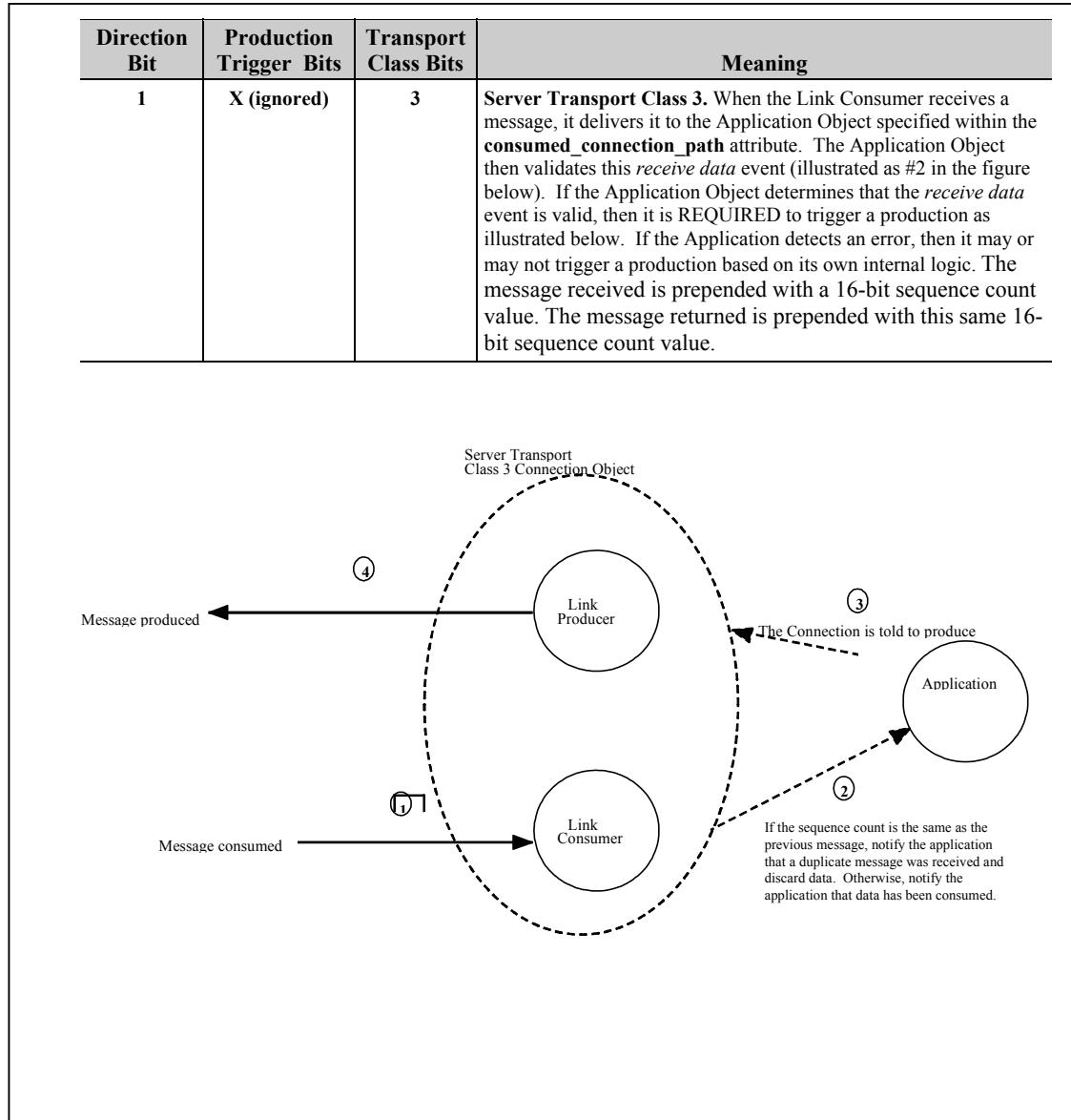
### 3-4.3.3.2 Server Transport Class 2 Behavior

**Figure 3-4.11. Server Transport Class 2 Connection Object**



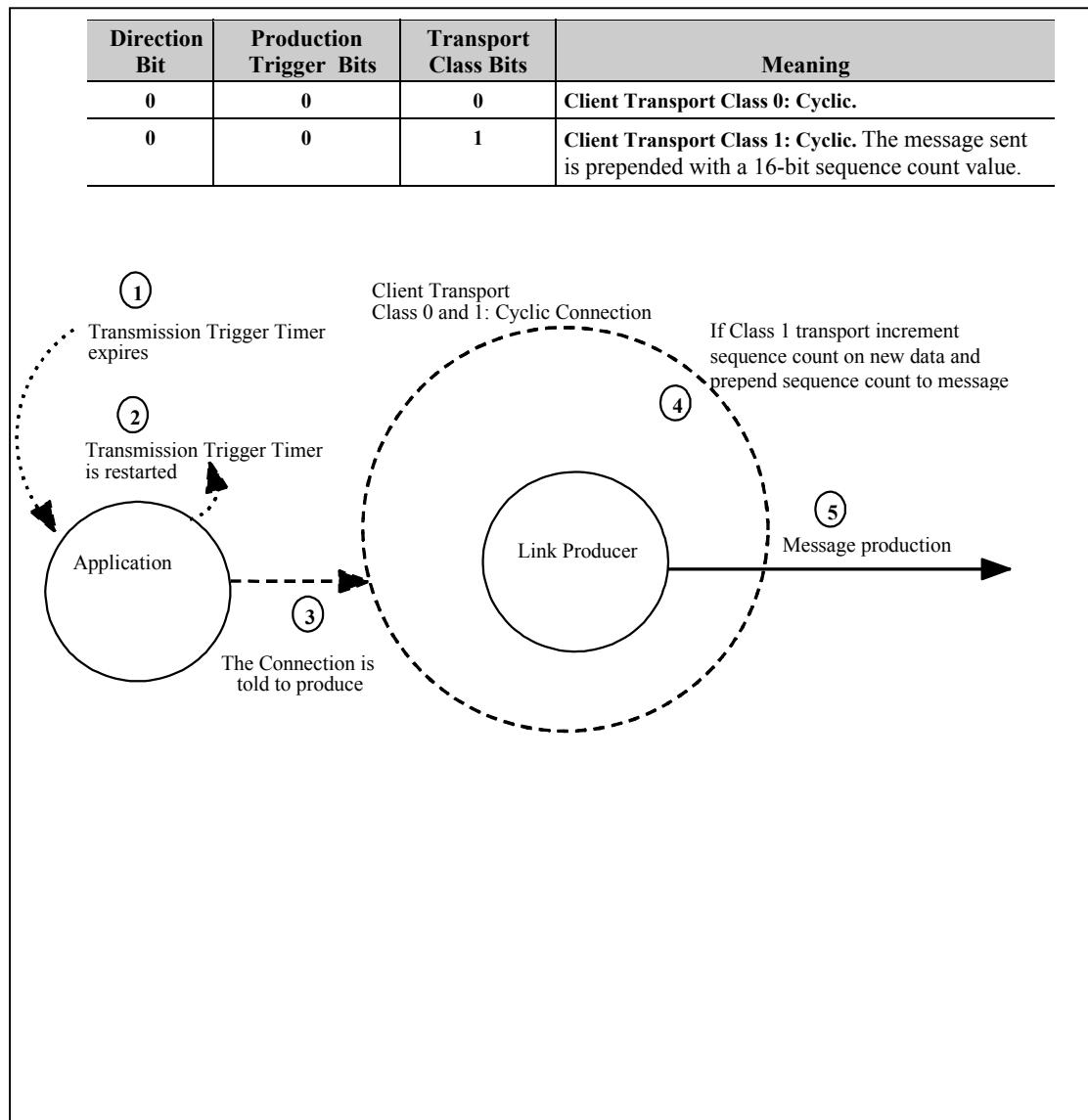
### 3-4.3.3.3 Server Transport Class 3 Behavior

**Figure 3-4.12. Server Transport Class 3 Behavior**



### 3-4.3.3.4 Client Transport Class 0 and 1 Behavior: Cyclic

**Figure 3-4.13. Client Transport Class 0 and 1 Behavior: Cyclic**

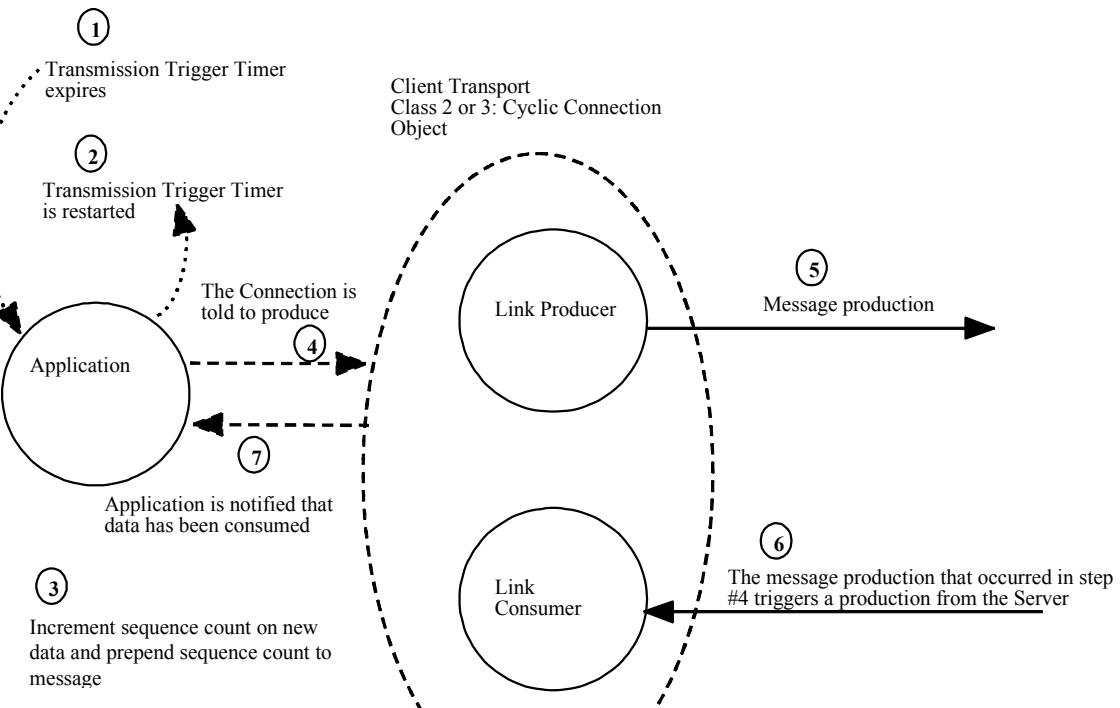


### 3-4.3.3.5 Client Transport Class 2 and 3 Behavior: Cyclic

**Figure 3-4.14. Client Transport Classes 2 & 3 Behavior: Cyclic**

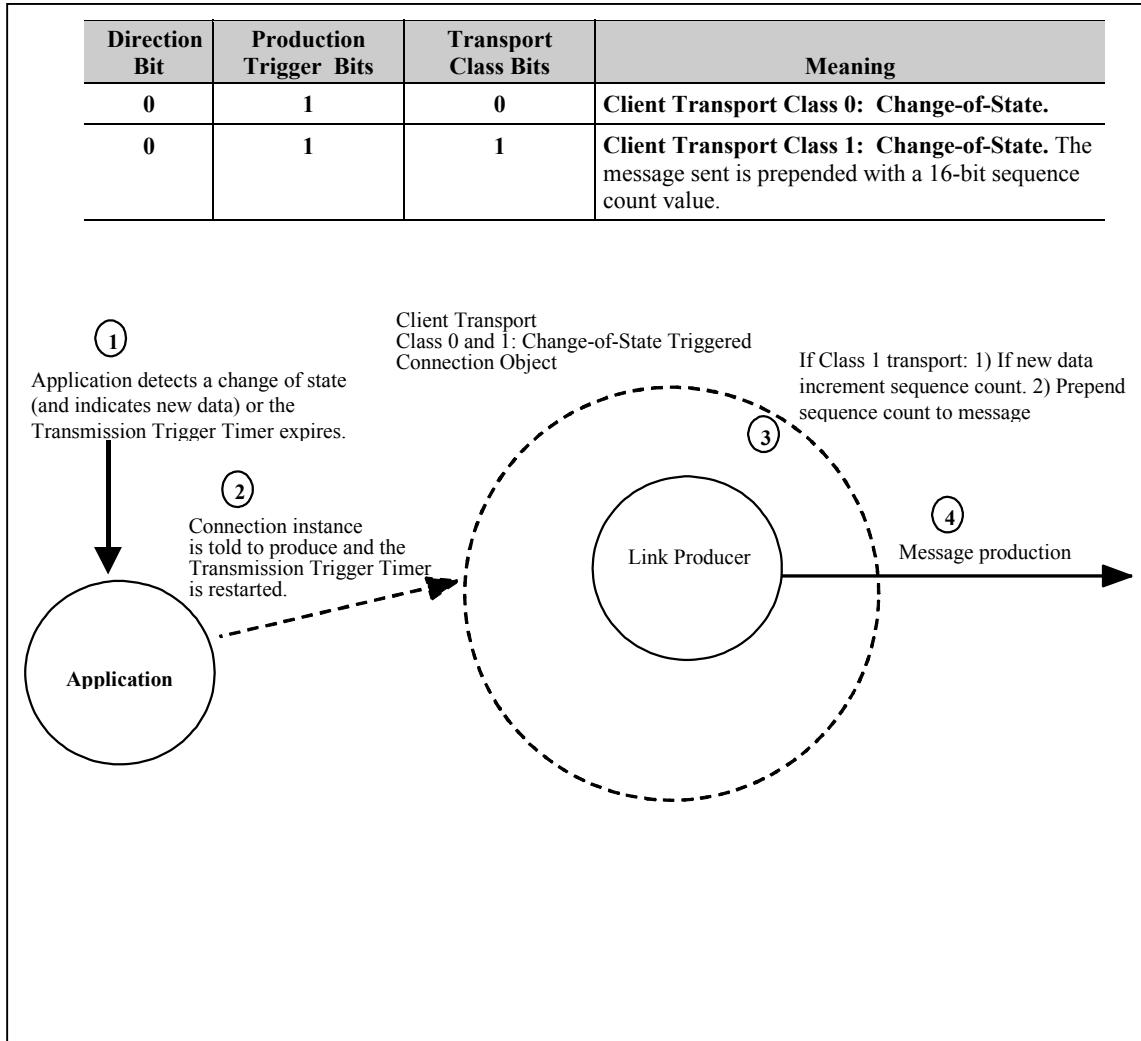
Direction Bit	Production Trigger Bits	Transport Class Bits	Meaning
0	0	2	Client Transport Class 2: Cyclic
0	0	3	Client Transport Class 3: Cyclic

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.



### 3-4.3.3.6 Client Transport Class 0 and 1 Behavior: Change-Of-State

**Figure 3-4.15. Client Transport Class 0 and 1 Behavior: Change-Of-State**

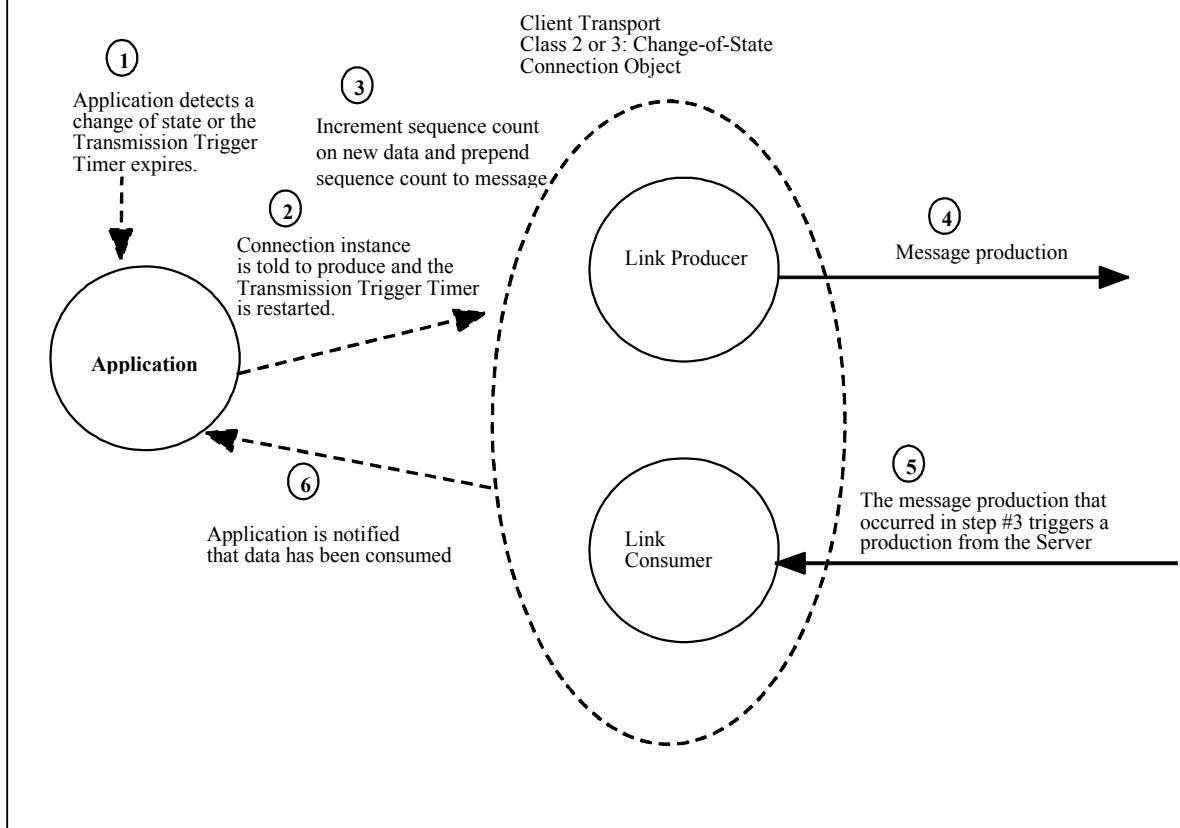


### 3-4.3.3.7 Client Transport Class 2 and 3 Behavior: Change-Of-State

**Figure 3-4.16. Client Transport Classes 2 & 3 Behavior: Change-Of-State**

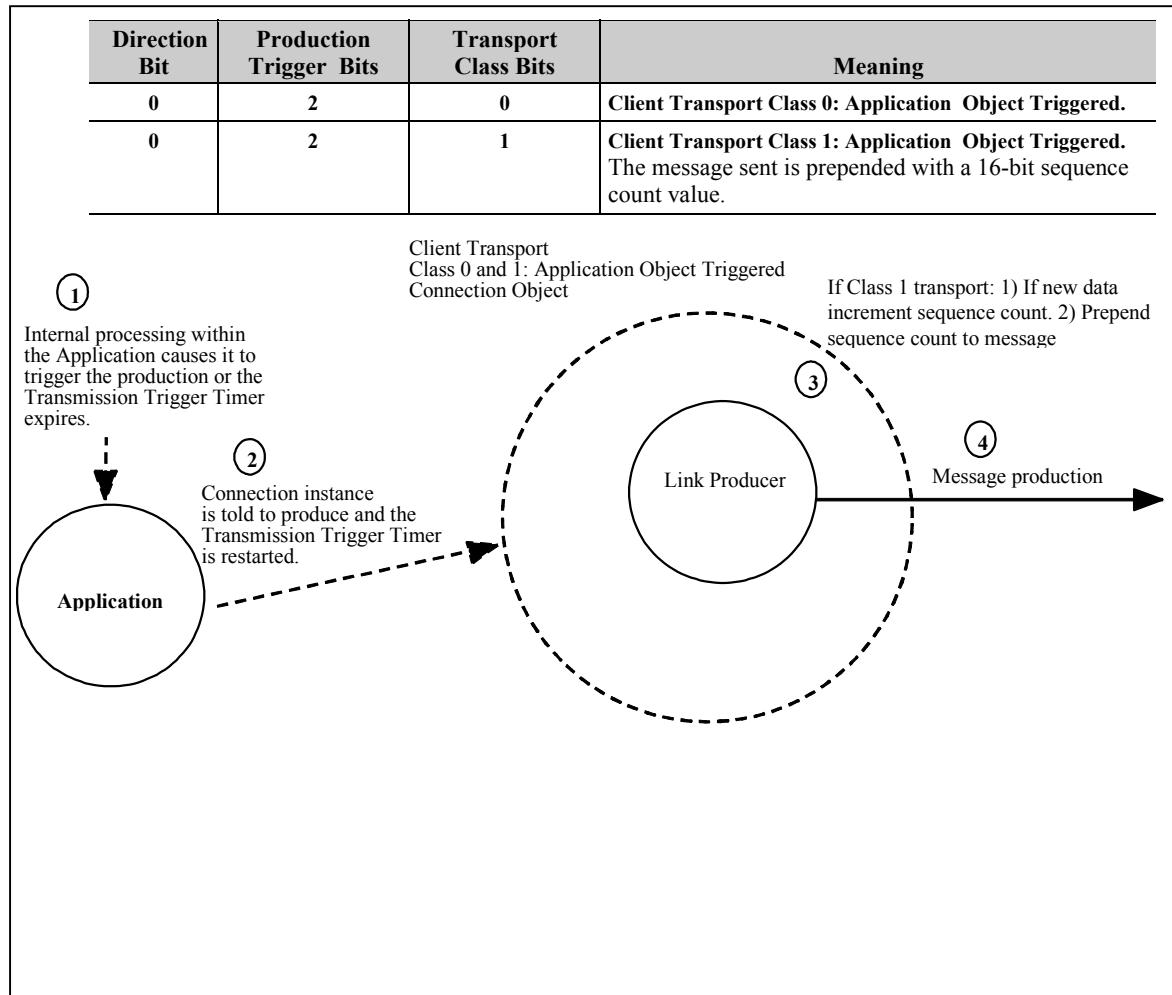
Direction Bit	Production Trigger Bits	Transport Class Bits	Meaning
0	1	2	Client Transport Class 2: Change-of-State
0	1	3	Client Transport Class 3: Change-of-State

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.



### 3-4.3.3.8 Client Transport Class 0 and 1 Behavior: Application Object Triggered

**Figure 3-4.17. Client Transport Class 0 and 1 Behavior: Application Object Triggered**

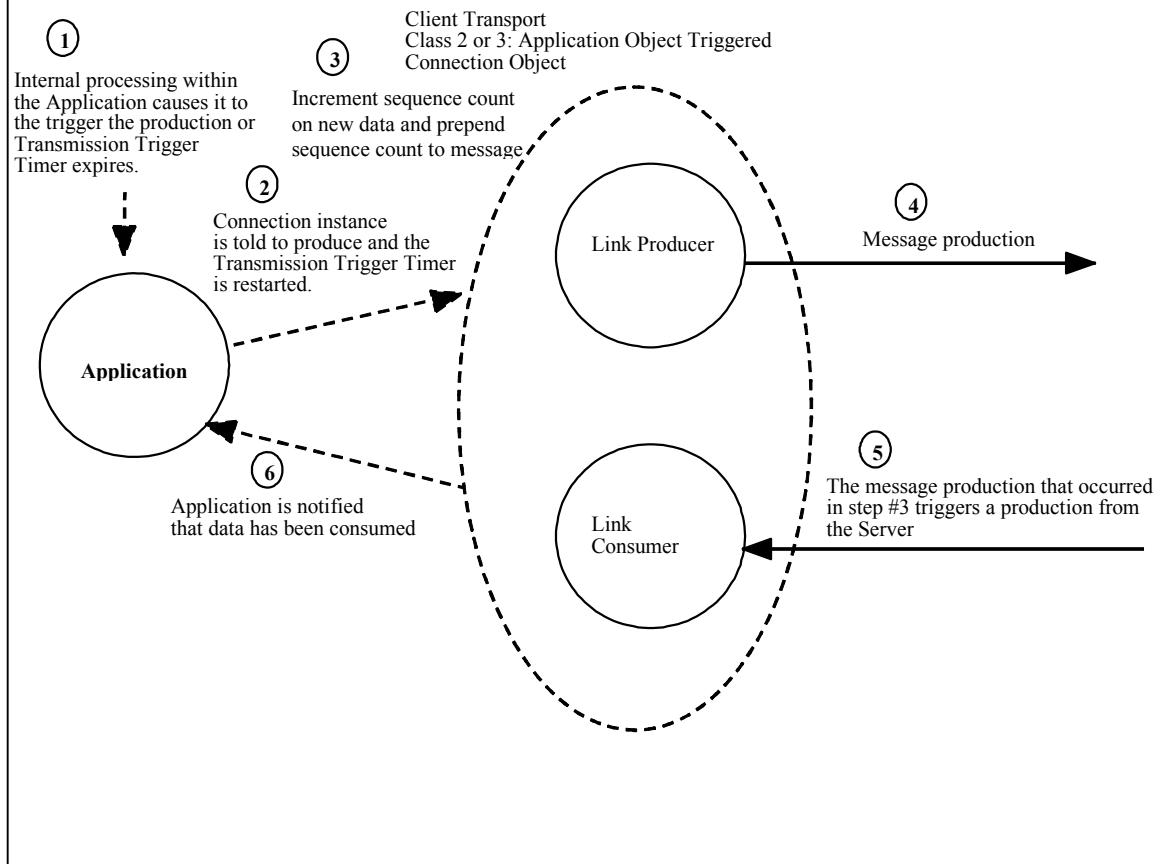


### 3-4.3.3.9 Client Transport Class 2 and 3 Behavior: Application Object Triggered

**Figure 3-4.18. Client Transport Classes 2 & 3 Behavior: Application Object Triggered**

Direction Bit	Production Trigger Bits	Transport Class Bits	Meaning
0	2	2	Client Transport Class 2: Application Object Triggered
0	2	3	Client Transport Class 3: Application Object Triggered

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.



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

<b>TransportClass_trigger bits</b>	<b>Meaning</b>
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 a 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 a Explicit Messaging Connection.
1 111 1111	Default value assigned to this attribute within an I/O Connection.

#### 3-4.3.4 **UINT DeviceNet\_produced\_connection\_id**

Contains the DeviceNet Connection ID to be associated with transmissions sent across this connection (if any). This is the value that will be specified in the CAN Identifier Field when this Connection transmits. See chapter 3, section 3-2, DeviceNet's Use of the CAN Identifier Field. This value is loaded directly into the associated Link Producer's **connection\_id** attribute. The following values are defined:

**Table 3-4.18. Values defined for the produced\_connection\_id attribute**

<b>Value</b>	<b>Meaning</b>
<b>0 - 7F0<sub>hex</sub></b>	The value to be placed in the CAN Identifier Field when this Connection transmits.
<b>800<sub>hex</sub> - FFFE<sub>hex</sub></b>	Reserved by CIP
<b>FFFF<sub>hex</sub></b>	Default value assigned to this attribute within an I/O Connection. This attribute will retain this value if this Connection instance is not producing any data (consumer only).

#### 3-4.3.5 **UINT DeviceNet\_consumed\_connection\_id**

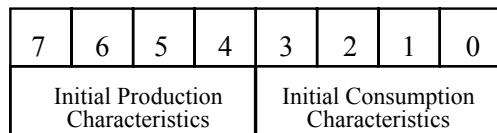
Contains the Connection ID, which identifies messages to be received across this connection (if any). This is the CAN Identifier Field value that is associated with messages this Connection Object receives. See chapter 3, section 3-2, DeviceNet's Use of the CAN Identifier Field. This value is loaded directly into the associated Link Consumer's **connection\_id** attribute. The following values are defined:

**Table 3-4.19. Values defined for the consumed\_connection\_id attribute**

Value	Meaning
0 - 7F0 <sub>hex</sub>	The value that identifies messages to be consumed. This will be specified in the CAN Identifier Field of messages that are to be consumed.
800 <sub>hex</sub> - FFF <sub>hex</sub>	Reserved by CIP
FFF <sub>hex</sub>	Default value assigned to this attribute within an I/O Connection. This attribute will retain this value if this Connection Instance is not consuming any data (producer only).

### 3-4.3.6 USINT DeviceNet\_initial\_comm\_characteristics

Defines the Message Group(s) across which productions and consumptions associated with this Connection occur. This byte is divided into two nibbles.

**Figure 3-4.20. DeviceNet\_initial\_comm\_characteristics attribute format**

The following table lists the values that are possible within the *Initial Production Characteristics* nibble (upper nibble) of the **initial\_comm\_characteristics** attribute.

**Table 3-4.21. Values for the Initial Production Characteristics Nibble**

Value	Meaning	
0	Produce across Message Group 1	The production associated with this Connection is to take place across Message Group 1. The producing module generates the Connection ID value and loads it into the Connection Object's produced_connection_id attribute. The producing module allocates a Message ID from its Group 1 Message ID pool and combines this with its Source MAC ID to generate the Connection ID. The numerically <u>lowest</u> available Group 1 Message ID is to be used in generating the produced_connection_id attribute value. This value must also be loaded into the corresponding consumed_connection_id attribute(s) associated with the consuming Connection Object(s).
1	Produce across Message Group 2 (Destination)	The production associated with this Connection is to take place across Message Group 2. Additionally, the intended recipient's MAC ID (Destination MAC ID) is to be placed within the MAC ID component of the Group 2 Identifier Field. <u>In this case, the consuming module generates the Connection ID value to be associated with transmissions across this connection.</u> When the consuming module has generated this value and loaded it into the appropriate Connection Object's consumed_connection_id attribute, it can be read and subsequently loaded into the producing Connection Object's produced_connection_id attribute.
2	Produce across Message Group 2 (Source)	The production associated with this Connection is to take place across Message Group 2. In addition, the producing module's MAC ID (Source MAC ID) is to be placed within the MAC ID component of the Group 2 Identifier. In this case, the producing module generates the Connection ID value and loads it into the Connection Object's produced_connection_id attribute. The numerically <u>lowest</u> available Group 2 Message ID is to be used in generating the produced_connection_id attribute value. This value must also be loaded into the corresponding consumed_connection_id attribute(s) associated with the consuming Connection Object(s).

Value	Meaning	
3	Produce across Message Group 3	The production associated with this Connection is to take place across Message Group 3. The producing module generates the Connection ID value and loads it into the Connection Object's produced_connection_id attribute. The producing module allocates a Message ID from its Group 3 Message ID pool and combines this with its Source MAC ID to generate the Connection ID. The numerically <u>lowest</u> available Group 3 Message ID is to be used in generating the produced_connection_id attribute value. This value must also be loaded into the corresponding consumed_connection_id attribute(s) associated with the consuming Connection Object(s).
4 - E	Reserved	
F	Default value	The default value assigned to the Initial Production Characteristics nibble within an I/O Connection. Note that if this is a <i>producing only</i> I/O Connection, then the default value remains in this nibble. Explicit Messaging Connection Objects automatically configure this attribute when the Connection is established.

Table 3-4.22 lists the possible values within the *Initial Consumption Characteristics* nibble (lower nibble) of the **DeviceNet\_initial\_comm\_characteristics** attribute.

**Table 3-4.22. Values for the Initial Consumption Characteristics Nibble**

Value	Meaning	
0	Consume a Group 1 Message	The message to be consumed will be transmitted across Message Group 1. The producing module generates the Connection ID value. This value must be loaded into the consumed_connection_id attribute associated with the consuming Connection Object(s).
1	Consume a Group 2 Message (Destination)	The message to be consumed will be transmitted across Message Group 2. The intended recipient's MAC ID (Destination MAC ID) is specified within the Group 2 Identifier. The consuming module generates the Connection ID value and loads it into the consumed_connection_id attribute associated with this Connection Object. The numerically <u>lowest</u> available Group 2 Message ID is to be used in generating the consumed_connection_id attribute value. This value must be loaded into the producing Connection Object's produced_connection_id attribute.
2	Consume a Group 2 Message (Source)	The message to be consumed will be transmitted across Message Group 2. The transmitting module's MAC ID (Source MAC ID) is specified within the Group 2 Identifier. In this case, the producing module generates the Connection ID value and loads it into the Connection Object's produced_connection_id attribute. This value must be loaded into the consumed_connection_id attribute associated with the consuming Connection Object(s).
3	Consume a Group 3 Message	The message to be consumed will be transmitted as a Group 3 Message. The producing module generates the Connection ID value. The Connection ID value must be loaded into this Connection Object's consumed_connection_id attribute.
4 - E	Reserved by CIP	
F	Default value	The default value assigned to the Initial Consumption Characteristics nibble within an I/O Connection. Note that if this is a <i>producing only</i> I/O Connection, then the default value remains in this nibble. Explicit Messaging Connections automatically configure this attribute when the Connection is established.

**Important:** The module that generates a Connection ID must guarantee that it does not allocate the Message ID/MAC ID pair in such a way that two separate modules are capable of transmitting identical bit patterns within the Identifier Field. Refer to section 3-4.8, Dynamic Management of Message IDs.

### 3-4.3.7 **UINT produced\_connection\_size**

The meaning of this attribute is different for Explicit Messaging Connections than it is for I/O Connections.

#### *For Explicit Messaging Connections:*

This attribute signifies the maximum number of **Message Body** bytes that a module is able to transmit across this Connection (the Message Body begins with the Service Field and ends with the last Service Specific data byte).

Modules that do not support the transmission of the Fragmentation Protocol initialize this attribute to the value 7 (Message Header (1 byte) + Message Body (7 bytes) = 8 bytes, which is the maximum length of a non-fragmented Explicit Message). Modules that cannot or do not predefine an up-front transmit limit load, place the value 0xffff into this attribute (there may still be a limit, however, it is not known in advance). Modules that support the Fragmentation Protocol, but place a known limit on the maximum amount of **Message Body** bytes that can be transmitted in a single fragmented series initialize this attribute accordingly.

**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.

#### *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 this attribute is set to a value greater than eight (8) in an I/O Connection, then the Connection will break up the data into multiple fragments.

**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.

### 3-4.3.8 **UINT consumed\_connection\_size**

The meaning of this attribute is different for Explicit Messaging Connections than it is for I/O Connections.

#### *For Explicit Messaging Connections:*

This attribute signifies the maximum number of **Message Body** bytes that this module is able to receive across this Connection (the Message Body begins with the Service Field and ends with the last Service Specific data byte).

Modules that do not support the reception of the Fragmentation Protocol initialize this to the value 7 (Message Header (1 byte) + Message Body (7 bytes) = 8 bytes, which is the maximum length of a non-fragmented Explicit Message). Modules that cannot or do not predefine an up-front receive limit load, place the value 0xffff into this attribute (there may still be a limit - it is just not known in advance). Modules that support the Fragmentation Protocol, but place a known limit on the maximum amount of **Message Body** bytes that can be received in a single fragmented series, initialize this attribute accordingly. 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 this attribute is set to a value greater than eight (8) in an I/O Connection, then the Connection will process the fragmentation protocol.

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.

#### **3-4.3.9    UINT expected\_packet\_rate**

This attribute is used to generate the values loaded into the *Transmission Trigger Timer* and the *Inactivity/Watchdog Timer*. See section 3-4.4, 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.

#### 3-4.3.10 **CIP\_produced\_connection\_id**

Contains the Connection ID which identifies messages to be sent across this connection (if any).

#### 3-4.3.11 **CIP\_consumed\_connection\_id**

Contains the Connection ID which identifies messages to be received across this connection (if any).

#### 3-4.3.12 **USINT watchdog\_timeout\_action**

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 3-4.23. Values for the **watchdog\_timeout\_action****

Value	Meaning
0	<i>Transition to Timed Out.</i> The Connection transitions to the <b>Timed Out</b> 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 <b>Established</b> 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 <b>Deferred</b> state if any child connection instances are in the <b>Established</b> state. If no child connection instances are in the <b>Established</b> 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

#### 3-4.3.13 **UINT produced\_connection\_path\_length**

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).

### 3-4.3.14 EPATH produced\_connection\_path

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.

### 3-4.3.15 UINT consumed\_connection\_path\_length

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).

### 3-4.3.16 EPATH consumed\_connection\_path

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.

### 3-4.3.17 UINT production\_inhibit\_time

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 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.

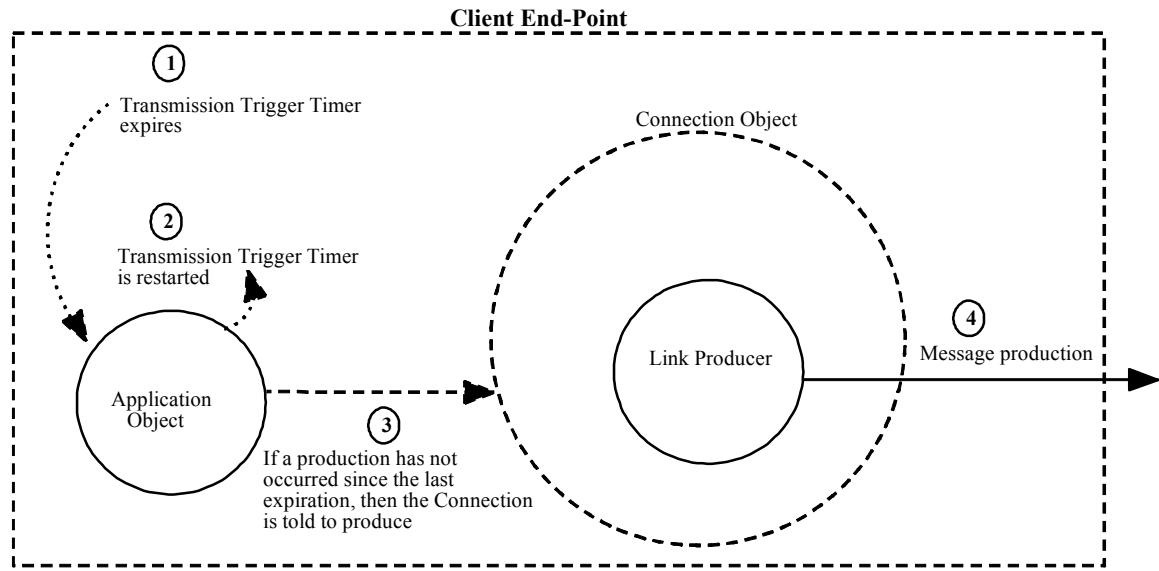
**Important:** For Explicit Messaging, the Application is responsible for providing *response timeout* facilities. The amount of time a Client waits for a Server to respond to a request depends on the application and, possibly, on the service. Due to Media Access mechanisms used for accessing the subnet,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

### 3-4.4.1 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 3-4.24. Transmission Trigger Timer**

As illustrated in Figure 3-4.24, 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.

#### 3-4.4.2 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

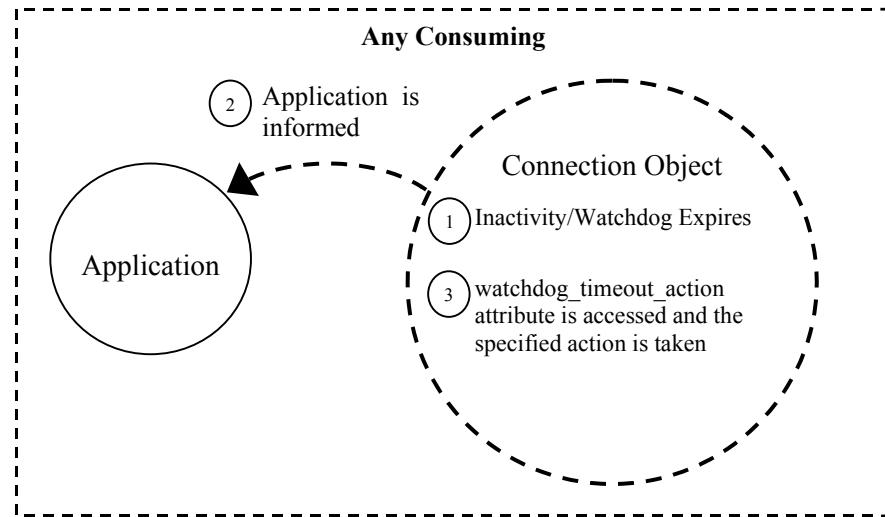
This timer is activated when the Connection transitions to the **Established** state. The tasks listed below are performed by a Connection immediately upon detecting that a **valid** message has been consumed:

- 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.

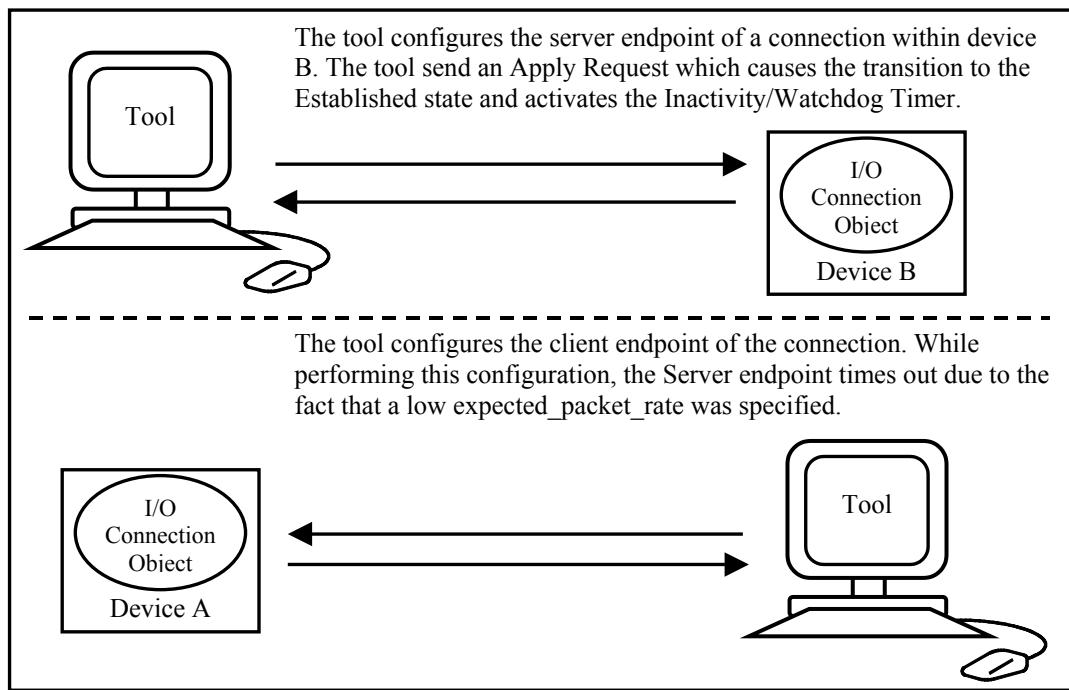
Expiration of this timer is an indication that the Connection Object has timed out while waiting to consume. The Connection Object performs the following steps when the Inactivity/Watchdog timer expires:

- issues an indication of this event to the application
- performs the action indicated by the **watchdog\_timeout\_action** attribute



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 4, depending on which value is numerically greater<sup>1</sup>. If the **expected\_packet\_rate** multiplied by 4 is greater than 10,000, then the **expected\_packet\_rate** multiplied by 4 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:



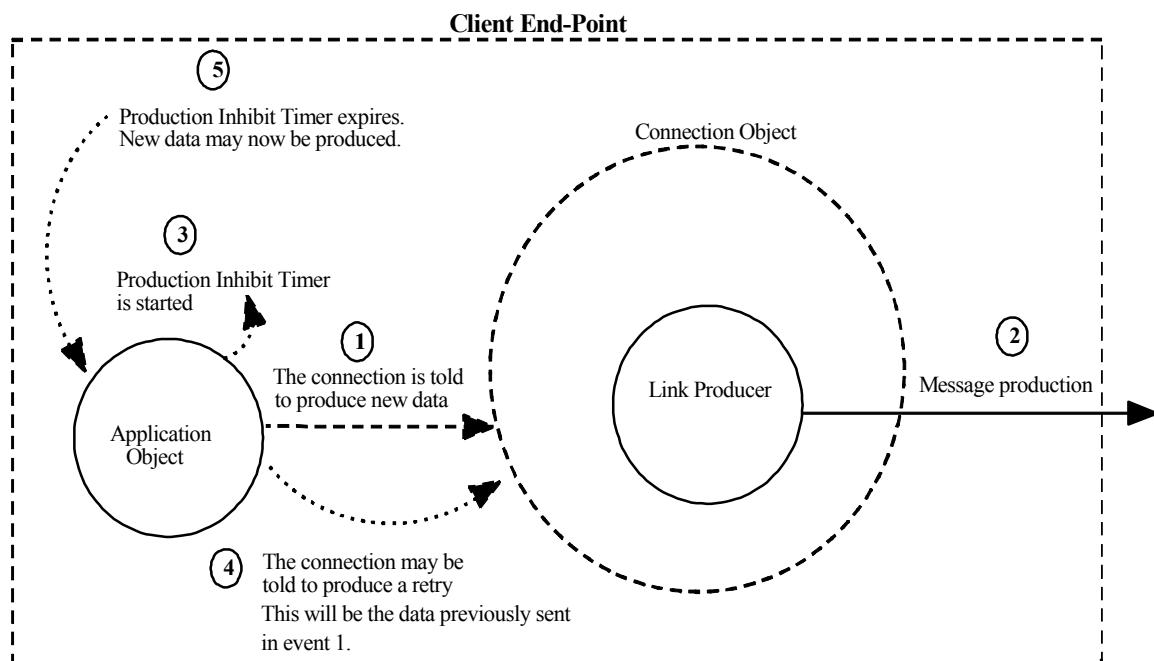
<sup>1</sup> If the **expected\_packet\_rate** attribute contains the value zero (0), then the Inactivity/Watchdog Timer is not activated and/or used by the Connection Object. A Connection Object whose **expected\_packet\_rate** attribute is zero (0) will never experience an Inactivity/Watchdog timeout.

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 4 ( $\text{expected\_packet\_rate} \ll 2$ ) as the number of milliseconds to load into the Inactivity/Watchdog Timer<sup>2</sup>.

### 3-4.4.3 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.



**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.

<sup>2</sup> 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.

### 3-4.5 Connection Object Instance Services

The Connection Object Instance supports the following CIP Common services:

**Table 3-4.24. Connection Object Instance Services**

Service Code	Need In Implementation	Service Name	Service Description
0E <sub>hex</sub>	Required	Get_Attribute_Single	Used to read a Connection Object attribute.
10 <sub>hex</sub>	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 3.21
05 <sub>hex</sub>	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.
09 <sub>hex</sub>	Optional	Delete	Used to delete a Connection Object and to release all associated resources.
0D <sub>hex</sub>	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 <i>Service Data</i> field of an Apply Attributes Response Message as indicated in Table 3.20. 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.

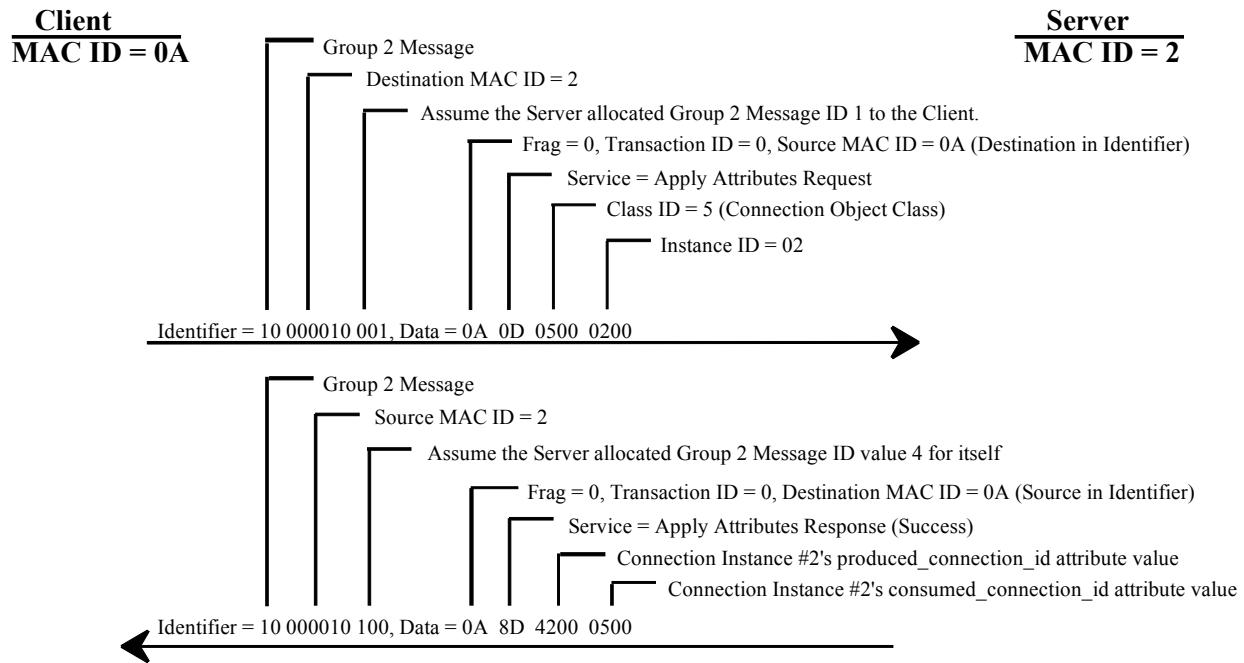
The following information is specified by a Connection Object Instance within the Service Data Field of a successful Apply\_Attributes response .

**Table 3-4.25. 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
Consumed Connection ID	UINT	Contains the value within the Connection Instance's <b>consumed_connection_id</b> attribute

The figure below illustrates the Apply\_Attributes service sent to a Connection Object Instance.

Assume an Explicit Messaging Connection was established across Group 2 and the Message Body Format was defined by the Server as DeviceNet (16/16).



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.

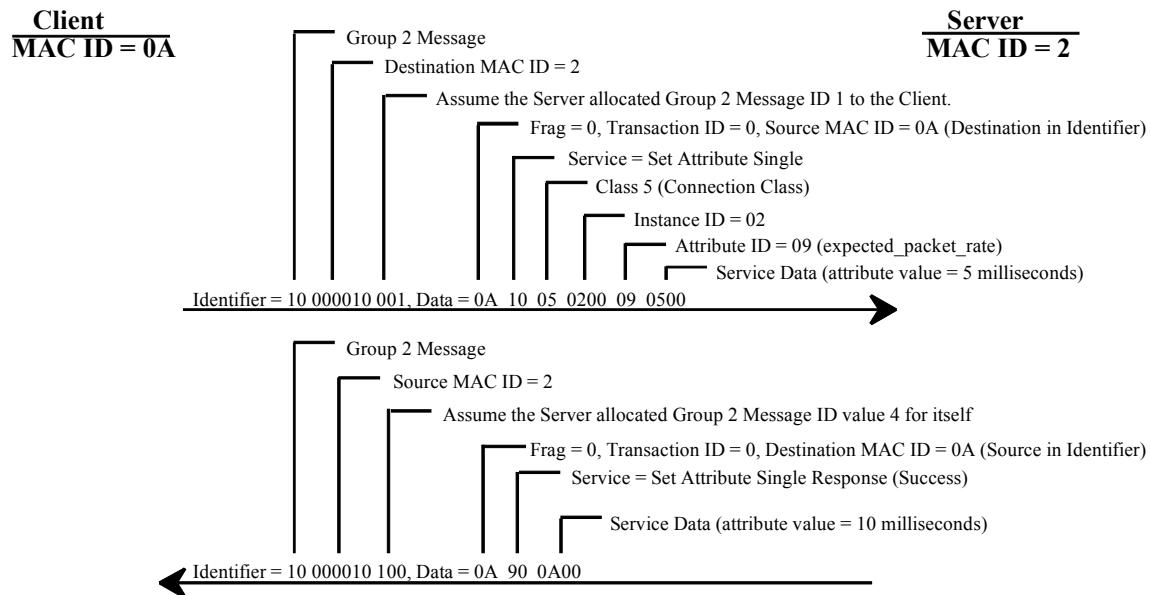
**Table 3-4.26. Service Data For Connection Object**

**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 below 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.

Assumes an 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.



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.

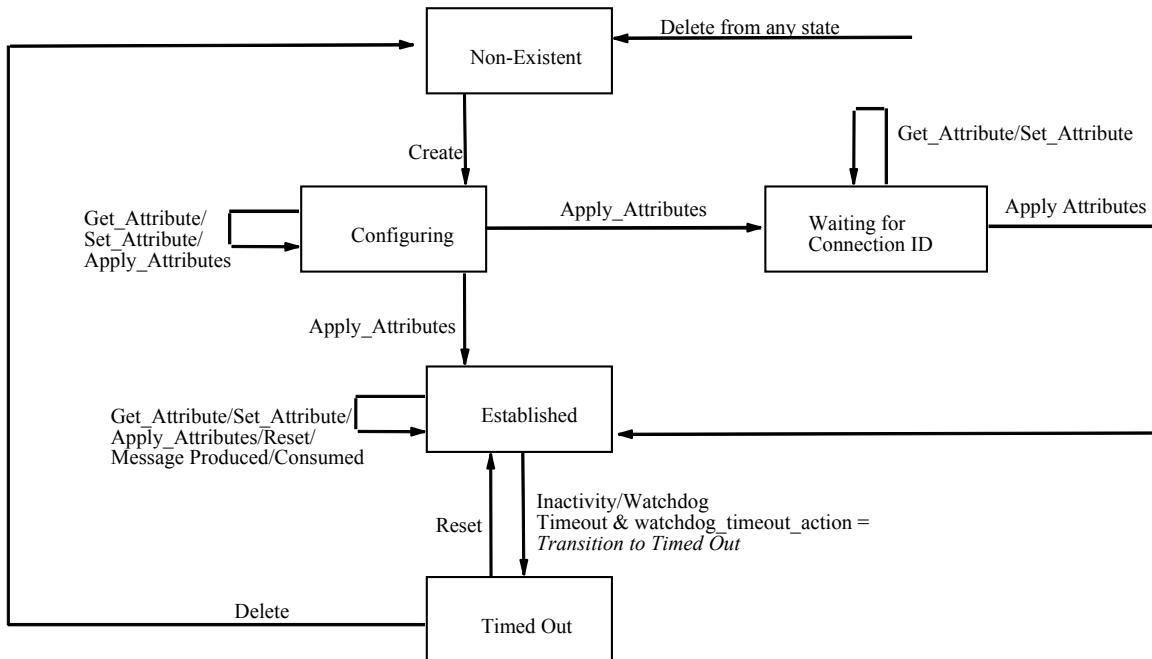
The Connection Object also supports the following internal services:

- **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.

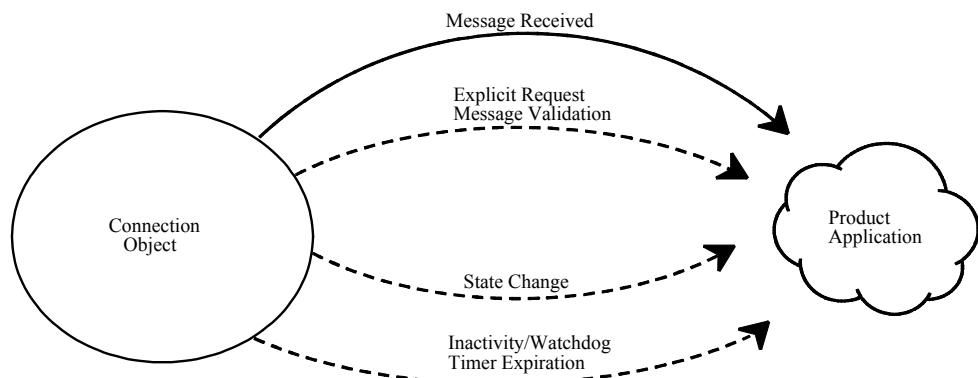
## 3-4.6 Connection Instance Behavior

### 3-4.6.1 I/O Connection Instance Behavior

Figure 3-4.27 provides a general overview of the behavior associated with an **I/O Connection Object** (`instance_type` attribute = I/O).

**Figure 3-4.27. I/O Connection Object State Transition Diagram**

A Connection Object issues the internal indications described in Figure 3-4.28 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 3-4.28. Internal Indications Issued by a Connection Object (Conceptual)**

**Important:** Table 3-4.29 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 3-4.29. 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 3-4.3).

**Table 3-4.29. 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 <b>instance_type</b> to I/O. Set all other attributes to default values. Transition to <b>Configuring</b> .	Not applicable	Not applicable	Not applicable	Not applicable
Connection Class receives a Delete Request	Error: Object does not exist (General Error Code 16 <sub>hex</sub> )	Release all associated resources. Transition to <b>Non-existent</b>	Release all associated resources. Transition to <b>Non-existent</b> .	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 16 <sub>hex</sub> )	Validate/service the request based on internal logic and per the Access Rules presented in section 3-4.7. 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 0C <sub>hex</sub> (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 3-4.7. Return appropriate response.	Validate/service the request based on internal logic and per the Access Rules presented in section 3-4.7. Return appropriate response.
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 3-4.7. Return response.	Validate/service the request based on internal logic and per the Access Rules presented in section 3-4.7. Return response.	Validate/service the request based on internal logic and per the Access Rules presented in section 3-4.7. Return response.	Validate/service the request based on internal logic and per the Access Rules presented in section 3-4.7. 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 16 <sub>hex</sub> )	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 = 0C <sub>hex</sub> )	Cancel the current Inactivity/Watchdog Timer. Using the value in the <b>expected_packet_rate</b> 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, <b>expected_packet_rate</b> = 00C <sub>hex</sub> ).	Using the value in the <b>expected_packet_rate</b> attribute, start the Inactivity/Watchdog timer <b>and transition back to the Established state</b> . If the <b>expected_packet_rate</b> attribute has been set to zero (0) while the Connection was in the <b>Timed Out</b> state, then just transition back to <b>Established</b> without activating an Inactivity/Watchdog Timer.
Apply_Attributes	Error: Object does not exist (General Error Code 16 <sub>hex</sub> )	Deliver Connection Object to the application which validates the attribute information. If either of the <b>connection_id</b> 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 <b>Waiting For Connection ID</b> 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 <b>Established</b> state. If an error is detected, then an Error Response is returned and the Connection remains in the <b>Configuring</b> 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 <b>connection_id</b> 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 <b>Waiting For Connection ID</b> state. The inability to generate a produced or consumed connection ID value IS NOT reported as an error. If the produced and/or consumed_ <b>connection_id</b> attributes are now validly configured, then transition to the <b>Established</b> state and return a Successful Response <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.	All modifications take place immediately once the Connection has transitioned to the <b>Established</b> state. Return Error: The object cannot perform the requested service in its current mode/state. (General Error Code value = 0C <sub>hex</sub> )	Error: The object cannot perform the requested service in its current mode/state. (General Error Code value = 0C <sub>hex</sub> )

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 3-4.3)</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 <code>expected_packet_rate</code> attribute is non-zero, restart the Transmission Trigger Timer.	Return internal error - do not send the message
Inactivity/Watchdog Timer expires	Not applicable	Not applicable	Not applicable	Examine the <code>watchdog_timeout_action</code> attribute of the Connection and perform the indicated action. If the <code>watchdog_timeout_action</code> attribute indicates that the Connection is to remain in the <b>Established</b> 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* (02<sub>hex</sub>) 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* (09<sub>hex</sub>) 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 CAN Identifier Field matches the `consumed_connection_id` attribute is received. If a message is received whose CAN Identifier Field 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 3.22 , then an Error Response specifying *Service Not Supported* (General Error Code 08) is returned.

### 3-4.6.2 CIP Bridged Connection Instance Behavior

Figure 3-4.29 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 3-4.29. CIP Bridged Connection Object State Transition Diagram**

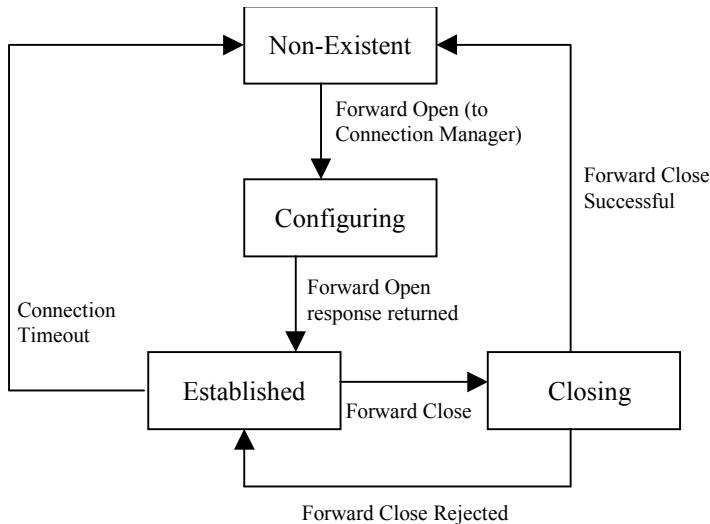


Table 3-4.30 provides a detailed State Event Matrix for a CIP Bridged Connection Object.

**Table 3-4.30. 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

Event	CIP Bridged Connection Object State			
	Non-Existent	Configuring	Established	Closing
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>
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 3-4.7. Return appropriate response.	Validate/service the request based on internal logic and per the Access Rules presented in section 3-4.7. Return appropriate response.	Validate/service the request based on internal logic and per the Access Rules presented in section 3-4.7. Return appropriate response.
Set_Attribute_Single	Error: Object does not exist (General Error Code 16 <sub>hex</sub> )			
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 <b>Non-Existent</b>	Release all resources and transition to <b>Non-Existent</b>

### 3-4.6.3 Explicit Messaging Connection Instance Behavior

Figure 3-4.31 provides a general overview of the behavior associated with an **Explicit Messaging Connection Object** (`instance_type` attribute = Explicit Messaging).

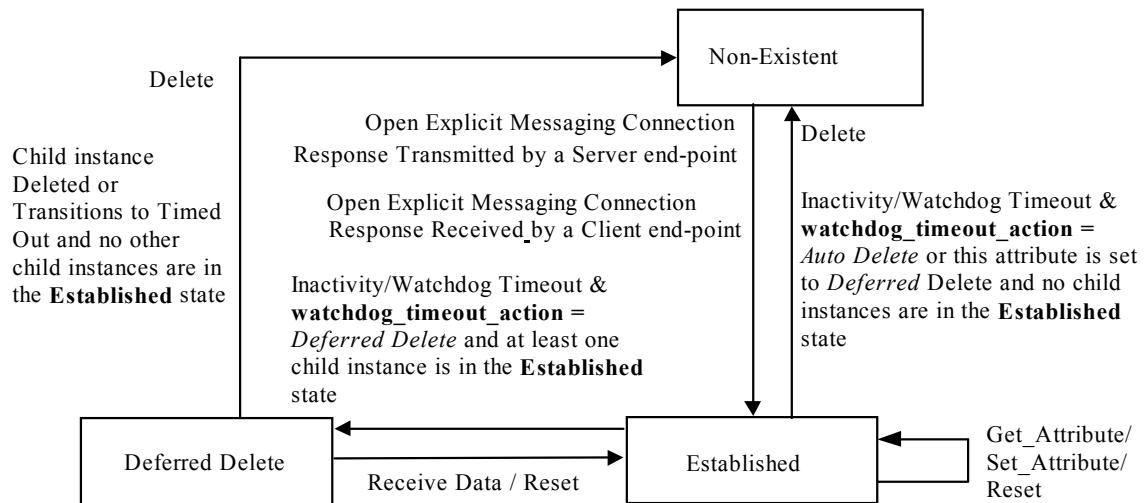
**Figure 3-4.31. Explicit Messaging Connection Object State Transition Diagram**

Table 3-4.32 provides a detailed State Event Matrix for an Explicit Messaging Connection Object. Implementations should be based on the information in Table 3-4.32.

**Table 3-4.32. 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 16 <sub>hex</sub> )	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 16 <sub>hex</sub> )	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 16 <sub>hex</sub> )	Validate/service the request based on internal logic and per the Access Rules presented in section 3-4.7. Return appropriate response.	Validate/service the request based on internal logic and per the Access Rules presented in section 3-4.7. Return appropriate response.
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 3-4.7. Return appropriate response.	Validate/service the request based on internal logic and per the Access Rules presented in section 3-4.7. Return appropriate response.

Event	Explicit Messaging Connection Object State		
	Non-Existent	Established	Deferred Delete
Reset	Error: Object does not exist (General Error Code 16 <sub>hex</sub> )	Cancel the current 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>	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 16 <sub>hex</sub> )	Error: The object cannot perform the requested service in its current mode/state. (General Error Code value = 0C <sub>hex</sub> )	Error: The object cannot perform the requested service in its current mode/state. (General Error Code value = 0C <sub>hex</sub> )
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> .

<sup>1</sup>If the configuration indicates that a connection resource needs to be allocated and an available resource does not exist, then an Error Response whose General Error Code indicates *Resource Unavailable* (02<sub>hex</sub>) is returned.

<sup>2</sup>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 whose 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 3-4.32, then an Error Response specifying *Service Not Supported* (General Error Code 08) is returned.

### 3-4.7 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 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 Error Code value = 0C<sub>hex</sub> - see Appendix B for more information on error codes).

**Table 3-4.33. 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

Attribute	I/O Connection State				
	Non-Existent	Configuring	Waiting for Connection ID	Established	Timed Out
<sup>1</sup> When a Connection Object is in the <b>Established</b> state, any modifications to the <b>expected_packet_rate</b> attribute 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> attribute:					
	- the current Inactivity/Watchdog Timer is canceled				
	- a new Inactivity/Watchdog Timer is activated based on the new value in the <b>expected_packet_rate</b> attribute.				

**Table 3-4.34. CIP Bridged Connection Object Attribute Access**

Attribute	Default Value <sup>1</sup>	CIP Bridged Connection State			
		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

<sup>1</sup>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 3-4.35. 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

<sup>1</sup>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.

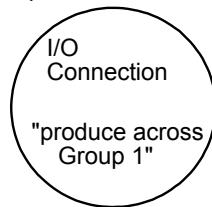
### 3-4.8 Dynamic Management Of Message IDs

Dynamically establishing both I/O and Explicit Messaging Connections requires the end-points go through an internal Message ID allocation process. For example:

1. Current state of the "Group 1 Message ID Allocation Table"

Message ID 0	Allocated
Message ID 1	Allocated
Message ID 2	Available
Message ID 3	Available
.....	.....
Message ID 0F	Available

2. An I/O Connection is created & configured which requests that a new production take place across Group 1



3. The next available Group 1 Message ID is allocated for use within the I/O Connection

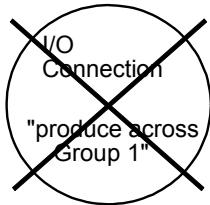
Message ID 0	Allocated
Message ID 1	Allocated
Message ID 2	Allocated
Message ID 3	Available
.....	.....
Message ID 0F	Available

Additionally, devices which implement fully dynamic Connection creation/deletion capabilities will also have to implement logic which marks a previously allocated Message ID as available when that Message ID is no longer in use. For example:

1. Current state of the "Group 1 Message ID Allocation Table"

Message ID 0	Allocated
Message ID 1	Allocated
Message ID 2	Allocated
Message ID 3	Available
.....	
Message ID 0F	Available

2. The I/O Connection is deleted

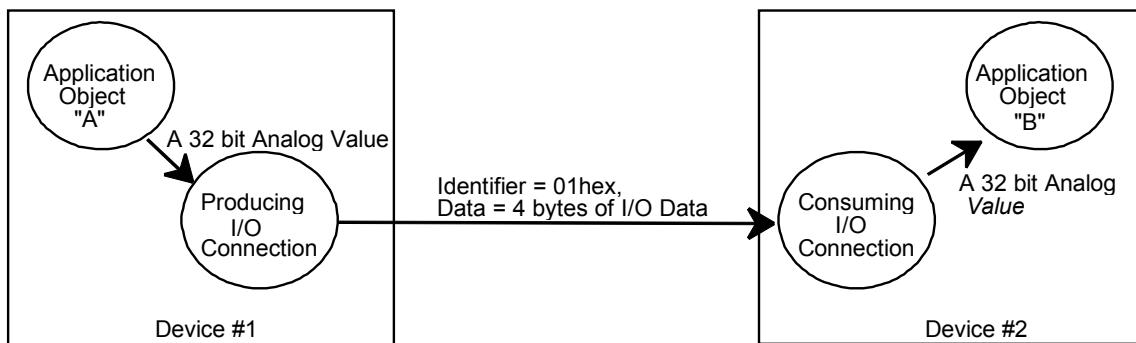


3. Group 1 Message ID value 2 is now available again.

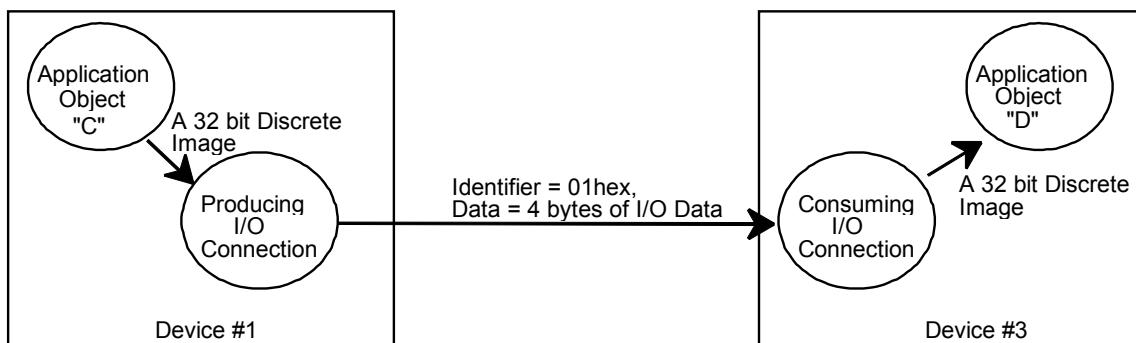
Message ID 0	Allocated
Message ID 1	Allocated
Message ID 2	Available
Message ID 3	Available
.....	
Message ID 0F	Available

The act of re-using a previously allocated Message ID for a different purpose gives rise to many issues. For example:

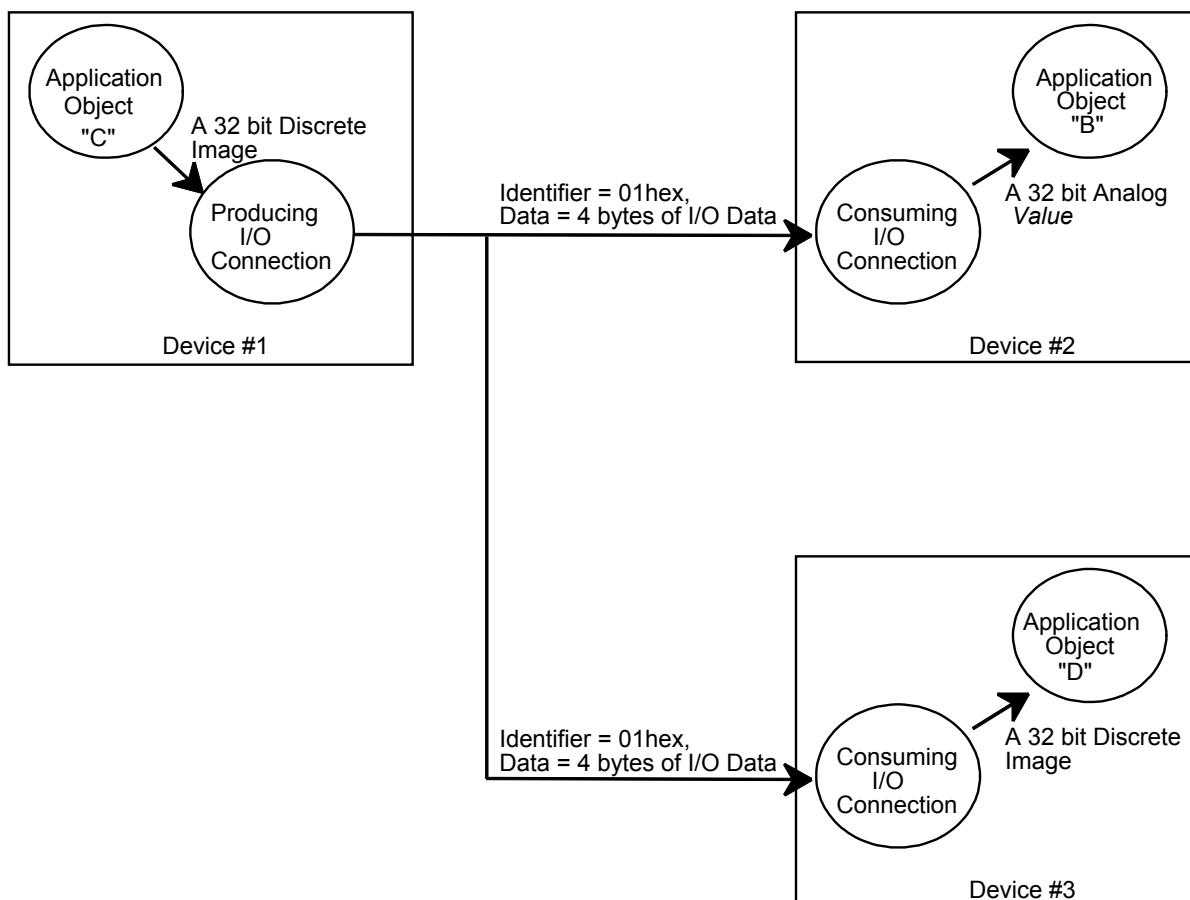
1. Assume the following configuration where the Producing I/O Connection in Device #1 is Analog Value identified by Identifier Field value 01hex. The Consuming I/O Connection is Identifier 01hex and when a consumption occurs it hands the data up to Application Object 'B'. Application Object 'B' assumes it is receiving an Analog Value and processes accordingly.



2. Assume that the Producing I/O Connection is Deleted and the associated Message ID is marked as available. Assume a new Producing I/O Connection is created and configured such that it re-uses this Message ID and generates the Identifier 01hex which is now associated with a 32 bit Discrete Image. Device #3 is configured to consume this message as illustrated.



3. There is a problem in that Application Object B within Device #2 processes a received I/O Message as a 32 bit Analog Value. The associated Consuming I/O Connection within Device #2 is still active and screening for Message ID 01hex. Application Object B in Device #2 will now receive what it believes to be an Analog Value but it is actually a 32 bit Discrete Image from a totally different source Application Object!



Not one single approach can resolve these Connection ID-related error scenarios in all situations/systems. To reduce the likelihood of Connection ID related error scenarios, the following implementation guidelines must be followed:

1. The allocation process must make every attempt to ensure that no two modules are capable of transmitting an identical bit pattern within the Connection Identifier Field.
2. If possible, the deallocation process must ensure that all end-points of a Connection have timed out prior to re-using a Message ID.

To further refine guideline #2, apply the following logic **when the decision has been made to mark a previously allocated Message ID as available**:

- If the Message ID is associated with a Connection that activates an Inactivity/Watchdog Timer, then a new Inactivity/Watchdog timer is activated. Upon expiration of this timer, the Message ID is marked as available.
- If the Message ID is associated with a Connection that does not activate an Inactivity/Watchdog Timer, then the assumption is made that this was taken into consideration when the Connection was established and the Message ID can be immediately marked as available.

**Important:** Realize that for a Connection using Transport Class 0 or a Connection whose `expected_packet_rate` attribute has been set to zero (0), guideline #2 cannot be met using logic based on that Connection Object alone. For example:

- If the Server end-point of a Transport Class 0 Connection experiences an Inactivity/Watchdog timeout, it cannot know the *state* of the Client based solely on this Connection. The Client could have *just missed* transmitting the message in a timely fashion and could still think the Connection is operating normally.
- If the Client end-point of a Connection whose `expected_packet_rate` has been set to zero (0) is deleted for some reason, the Server end-point(s) could still be active.

For the types of connections discussed in the bullet list above, use caution when performing tasks that will result in the deallocation and possible re-use of the associated Message ID(s). (i.e., configuring the `watchdog_timeout_action` attribute to *Auto Delete*, manually Deleting a Connection Object via the transmission of the Delete service)

**3-5****CONNECTION MANAGER OBJECT CLASS DEFINITION****Class ID Code: 0x6**

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
01 <sub>hex</sub>	Optional	Get_Attribute_All	Returns the contents of all attributes of the class.
0E <sub>hex</sub>	Conditional	Get_Attribute_Single	Used to read a Connection Manager Class attribute value. This service is Required if any of the Connection Manager Class Attributes are 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:

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 3-5.3. Connection Manager Object Instance Attributes**

<b>Attr ID</b>	<b>Need In Implementation</b>	<b>Access Rule</b>	<b>Attribute Name</b>	<b>Data Type</b>	<b>Description of Attribute</b>	<b>Semantics</b>
1	Optional	Set	Open Requests	UINT	Number of Forward Open service requests received.	
2	Optional	Set	Open Format Rejects	UINT	Number of Forward Open service requests which were rejected due to bad format.	
3	Optional	Set	Open Resource Rejects	UINT	Number of Forward Open service requests which were rejected due to lack of resources.	
4	Optional	Set	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	Close Requests	UINT	Number of Forward Close service requests received.	
6	Optional	Set	Close Format Requests	UINT	Number of Forward Close service requests which were rejected due to bad format.	
7	Optional	Set	Close Other Requests	UINT	Number of Forward Open service requests which were rejected for reasons other than bad format.	
8	Optional	Set	Connection Timeouts	UINT	Number of connection timeouts which have occurred.	
9	Optional	Get	Connection Entry List	STRUCT of	Defines timing associated with this Connection	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.
			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	CPU_Utilization <sup>1</sup>	UINT	CPU Utilization in tenths of a percent.	Range of 0 - 1000 representing 0 to 100%.
12	Optional	Get	MaxBuffSize <sup>1</sup>	UDINT	Amount of buffer space originally available.	Size in bytes
13	Optional	Get	BufSize Remaining <sup>1</sup>	UDINT	Amount of buffer space available at this time.	Size in bytes

<sup>1</sup> 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 3-5.4. Connection Manager Object Instance Common Services**

Service Code	Need In Implementation	Service Name	Service Description
01 <sub>hex</sub>	Optional	Get_Attribute_All	Returns the contents of all attributes of the class.
0E <sub>hex</sub>	Conditional	Get_Attribute_Single	Used to read a Connection Manager Objectinstance attribute. This service is Required if any of the Connection Manager Instance Attributes are supported.
10 <sub>hex</sub>	Optional	Set_Attribute_Single	Used to modify a Connection Manager Object instance attribute.

#### 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.

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)							
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)							

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
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 3-5.5. 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
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>	N/A	Ex_Forward_Open	Reserved for definition of connection opens with a size larger than 504 bytes.
5A <sub>hex</sub>	Conditional	Get_Connection_Owner	Determine the owner of a redundant connection

### 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 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 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.

Network connection parameters shall be provided as a single 16-bit word that contains the fields in the following figure:

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)										

- *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.
- *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 *quantised* 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 quantised 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 3-5.6 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 3-5.7. 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 can not 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 Vendor ID**

The vendor number shall be a unique number assigned to the various vendors of products. Each vendor has a unique number assigned.

#### **3-5.5.1.6 Originator Serial Number**

The originator serial number shall be a unique 32-bit value that is assigned to a device at the time of manufacture. This value shall be guaranteed to be unique for all devices manufactured by the same vendor. No significance shall be attached to the number. The combination of Vendor ID and Originator Serial Number shall be unique throughout the entire system.

#### **3-5.5.1.7 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.

### 3-5.5.1.8 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.

### 3-5.5.1.9 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 3-5.8. Encoded Application Path Ordering**

Network Connection Parameters		Data Segment Present	Number of Encoded Application Paths		
O to T Connection Type	T to O Connection Type		1	2	3
Null	Null	Yes	Path is for Configuration.	Invalid	Invalid
		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.	Invalid	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.	Invalid

### 3-5.5.1.10 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;

### 3-5.5.1.11 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

The Forward Open Service (Service Code =  $54_{hex}$ ) is used to establish a Connection with a Target Device. This service results in local connection establishment on each link along the path. 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 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  $O \Rightarrow T$  and  $T \Rightarrow O$  designations;  $O \Rightarrow T$  means originator to target, and  $T \Rightarrow O$  means target to originator.

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 3-5.9. Forward Open Request**

Parameter Name	Data Type	Description
Priority/Time_tick	BYTE	Used to calculate request timeout information. See below.
Time-out_ticks	USINT	Used to calculate request timeout information. See below.
O_to_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_to_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_to_T RPI	UDINT	Originator to Target requested packet rate, in microseconds.
O_to_T Network Connection Parameters	WORD	See <b>Object Specific Service Parameters</b> above.
T_to_O RPI	UDINT	Target to Originator requested packet rate, in microseconds.
T_to_O Network Connection Parameters	WORD	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.

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 3-5.10. Successful Forward Open Response**

Parameter Name	Data Type	Description
O_to_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_to_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_to_T API	UDINT	Actual packet rate, originator to target. A router shall use the lesser of this value and the T_to_O API for the expected_packet_rate of the connection.
T_to_O API	UDINT	Actual packet rate, target to originator. A router shall use the lesser of this value and the O_to_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 3-5.11. Unsuccessful 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.

### 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 3-5.12. 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 3-5.13. 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 3-5.14. 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	

### 3-5.5.4 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 3-5.15. 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.
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.

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 3-5.16. Successful Unconnected Send Response**

Parameter Name	Data Type	Description
Reply Service	USINT	Reply service code.
Reserved	USINT	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 3-5.17 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. This list entails the ONLY Routing Errors that a CIP Router can return.

**Table 3-5.18. Routing Error Values**

General Status	Additional Status	Usage
01	0204 <sub>hex</sub>	Timeout indicator. Returned under the following circumstances: <ul style="list-style-type: none"> <li>Failure to establish an Explicit Messaging Connection.</li> <li>Timeout event occurs while waiting for an Explicit Messaging Response.</li> <li>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).</li> </ul>
01	0311 <sub>hex</sub>	Invalid Port ID specified in the <i>Route_Path</i> field.
01	0312 <sub>hex</sub>	Invalid Node Address specified in the <i>Route_Path</i> field.
01	0315 <sub>hex</sub>	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 3-5.19. Get Connection Data Service Request**

Parameter Name	Data Type	Description
Connection Number	UINT	Connection number

**Table 3-5.20. Get Connection Data Service Response**

Parameter Name	Data Type	Description
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 O2T CID	UDINT	
Target O2T CID	UDINT	
Connection Timeout Multiplier	USINT	
Reserved	USINT USINT USINT	
Originator RPI O2T	UDINT	
Originator API O2T	UDINT	
Originator T2O CID	UDINT	
Target T2O CID	UDINT	
Connection Timeout Multiplier	USINT	
Reserved	USINT USINT USINT	
Originator RPI T2O	UDINT	
Originator API T2O	UDINT	

### 3-5.5.6 Search Connection Data

The Unconnected\_Send service shall allow an application to send a message to a device

**Table 3-5.21. 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.

## 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 3-5.22. Connection Manager service request error codes**

General Status	Extended Status	Explanation
0x00		Service completed successfully.
0x01	0x0100	Connection in Use or Duplicate Forward Open.
0x01	0x0103	Transport Class and Trigger combination not supported
0x01	0x0106	Ownership Conflict
0x01	0x0107	Connection not found at target application.
0x01	0x0108	Invalid Connection Type. Indicates a problem with either the Connection Type or Priority of the Connection.
0x01	0x0109	Invalid Connection Size
0x01	0x0110	Device not configured
0x01	0x0111	RPI not supported. May also indicate problem with connection time-out multiplier, or production inhibit time.
0x01	0x0113	Connection Manager cannot support any more connections
0x01	0x0114	Either the Vendor Id or the Product Code in the key segment did not match the device
0x01	0x0115	Product Type in the key segment did not match the device
0x01	0x0116	Major or Minor Revision information in the key segment did not match the device
0x01	0x0117	Invalid Connection Point
0x01	0x0118	Invalid Configuration Format
0x01	0x0119	Connection request fails since there is no controlling connection currently open.
0x01	0x011A	Target Application cannot support any more connections
0x01	0x011B	RPI is smaller than the Production Inhibit Time.
0x01	0x0203	Connection cannot be closed since the connection has timed out
0x01	0x0204	Unconnected Send timed out waiting for a response.
0x01	0x0205	Parameter Error in Unconnected Send Service
0x01	0x0206	Message too large for Unconnected message service
0x01	0x0207	Unconnected acknowledge without reply
0x01	0x0301	No buffer memory available
0x01	0x0302	Network Bandwidth not available for data
0x01	0x0303	No Tag filters available
0x01	0x0304	Not Configured to send real-time data
0x01	0x0311	Port specified in Port Segment Not Available
0x01	0x0312	Link Address specified in Port Segment Not Available
0x01	0x0315	Invalid Segment Type or Segment Value in Path
0x01	0x0316	Path and Connection not equal in close
0x01	0x0317	Either Segment not present or Encoded Value in Network Segment is invalid.
0x01	0x0318	Link Address to Self Invalid
0x01	0x0319	Resources on Secondary Unavailable
0x01	0x031A	Connection already established
0x01	0x031B	Direct connection already established
0x01	0x031C	Miscellaneous
0x01	0x031D	Redundant connection mismatch

General Status	Extended Status	Explanation
0x01	0x031E	No more consumer resources available in the producing module
0x01	0x031F	No connection resources exist for target path
0x01	0x320 – 0x7FF	Vendor specific
0x02	n/a	Connection Manager resources are unavailable to handle service request
0x03	n/a	Invalid connection number specified by the Get_Connection_Data service. This is also returned by the Search_Connection_Data service if the specified connection is not found.
0x04	Zero Based Word Offset	Segment Type in path is invalid. The Extended Status shall be the word offset (0 based) to the word in the path where the error occurred. The offset starts at the first word after the path size. This error shall not be returned if an error occurs when parsing the Connection Path.
0x05	Zero Based Word Offset	Destination in path is invalid. The Extended Status shall be the word offset (0 based) to the word in the path where the error occurred. The offset starts at the first word after the path size. This error shall not be returned if an error occurs when parsing the Connection Path.
0x07	n/a	Connection has been lost. This is used by the Get/Set Services when they are made through a connection.
0x08	n/a	Connection Manager does not support the requested Service.
0x09	Index to Element	Error in Data Segment. Extended Status shall be index to where the error was encountered in the Data Segment. The Configuration Revision Number if present in the Data Segment shall always be index 1. If the error occurs with the Get/Set Services, then the extended status indicates the attribute number that failed.
0x0C	Optional	Service cannot be performed while <b>Object</b> is in current state. The 1st word of Extended Status may optionally contain the object's current state.
0x10	Optional	Service cannot be performed while <b>Device</b> is in current state. The 1st word of Extended Status may optionally contain the device's current state.
0x11	n/a	Response data too large. This is used by the get services to indicate the amount of data requested was too large to fit into the response buffer.
0x13	n/a	Not enough data was received.
0x14	Attribute Id	Attribute specified in FIND service is not supported by Connection Manager
0x15	n/a	Too much data was received.
0x25	0x0114	Either the Vendor Id or the Product Code in the key segment did not match the device. Used if the Key Segment was contained in the path.
0x25	0x0115	Product Type in the key segment did not match the device. Used if the Key Segment was contained in the path.
0x25	0x0116	Major or Minor Revision information in the key segment did not match the device. Used if the Key Segment was contained in the path.
0x26	n/a	Invalid path size

NOTE: The word “n/a” in the Extended Status Column is used to signify that there is no additional Extended Status which is required to be returned for the particular General Status Code.

The word “optional” in the Extended Status Column is used to signify that if Extended Status information is used, then the first word of that extended status is already defined and user defined extended status shall begin with the second word of extended status.

### 3-5.6.2 General Status Code 0x01

#### 3-5.6.2.1 Usage

This general status code shall be returned if there is a problem with a parameter in a service request to the connection manager. The following sections shall provide details of the specific extended status codes that are to be returned with the general status.

#### 3-5.6.2.2 Connection in use (extended status = 0x0100)

This extended status code shall be returned when an originator is trying to make a connection to a connection point with which the originator may have already established a connection (duplicate Forward\_Open). This may result from the originator establishing a connection with a connection point and the success response being lost in the return path. The originator shall then time-out on the request and attempt to establish the connection again even though the target actually established the connection.

The suggested response from the originator in this case shall be to either close and establish the connection again 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.

A duplicate forward open shall be defined as an open request that has the same connection serial number, vendor ID, and originator serial number as a connection on the target that is already open. If an intermediate node receives a duplicate Forward\_Open request, then the intermediate node shall pass on the request. However, the intermediate node shall not need to allocate transport resources for a new connection.

#### 3-5.6.2.3 Transport/trigger not supported (extended status = 0x0103)

A transport class and trigger combination has been specified which is not supported by the application. Although routers may use the transport/trigger field, they shall not fail the connection based on its contents. Only targets shall return this extended status code.

#### 3-5.6.2.4 Ownership conflict (extended status = 0x0106)

The connection cannot be established since another connection already "owns" some of the resources required for this connection. An example of this would be that only one connection can control an output point on an I/O Module. If a second controlling connection is attempted, this error shall be returned. This extended status code shall only be returned by a target node.

#### 3-5.6.2.5 Connection not found at target application (extended status = 0x0107)

This extended status code shall be returned by the close connection request, where the connection which is to be closed is not active at the target node. This extended status code shall only be returned by a target node. An intermediate node 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.

**3-5.6.2.6 Invalid connection type (extended status = 0x0108)**

This extended status code shall be returned as the result of specifying a connection type (including fixed/variable sized connection) or connection priority which is not supported by the target application. Only a target node shall return this extended status code.

**3-5.6.2.7 Invalid connection size (extended status = 0x0109)**

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 an intermediate device if the requested size is too large for the specified network.

**3-5.6.2.8 Device not configured (extended status = 0x0110)**

This extended status code shall be returned when a connection is requested from 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.

**3-5.6.2.9 RPI not supported (extended status = 0x0111)**

This extended status code shall be returned if the device can not support the requested O $\Rightarrow$ T or T $\Rightarrow$ 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.

**3-5.6.2.10 Connection Manager out of connections (extended status = 0x0113)**

The maximum number of connections allowed by this instance of the Connection Manager has been exceeded.

**3-5.6.2.11 Vendor Id or Product Code error (extended status = 0x0114)**

The Product Code or Vendor Id specified in the electronic key logical segment does not match the Product Code or Vendor Id in the target device.

**3-5.6.2.12 Product Type error (extended status = 0x0115)**

The Product Type specified in the electronic key logical segment does not match the Product Type in the target device.

**3-5.6.2.13 Revision mismatch (extended status = 0x0116)**

The major and minor revision information in conjunction with the exact match bit specified in the electronic key logical segment does not correspond to a valid revision for this device.

**3-5.6.2.14 Invalid connection point (extended status = 0x0117)**

The connection point specified in the connection path does not correspond to a valid connection point for the target application. This error could also be returned if a connection point was required, but not provided by a connection request.

**3-5.6.2.15 Invalid configuration format (extended status = 0x0118)**

An instance number specified for the configuration data does not correspond to a configuration instance. For example the connection path specifies a float configuration when the connection points specify a raw connection. This extended status code shall be used when the logical instance segment in the connection path is used to indicate a configuration type instead of an instance.

**3-5.6.2.16 Controlling connection not open (extended status = 0x0119)**

The extended status code shall be returned when an attempt is made to establish an echo (i.e. listen only) connection to a connection which has no controlling connection (i.e. owner).

**3-5.6.2.17 Application out of connections (extended status = 0x011A)**

The maximum number of connections supported by this instance of the Target Application has been exceeded.

For example, the Connection Manager could support 20 connections while the Target Application can only support 10 connections (there may be other Target Applications that an originator can connect to). On the 11th Connection Request to the Target Application, this extended status code would be used to signify that the Target Application is out of Connections.

**3-5.6.2.18 Connection timed out (extended status = 0x0203)**

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 at the originating node.

**3-5.6.2.19 Unconnected send timed out (extended status = 0x0204)**

The Unconnected Send Timed Out 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 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.

**3-5.6.2.20 Parameter error in unconnected send service (extended status = 0x0205)**

One of the parameters in the unconnected send service was in error.

This shall be caused by an invalid Connection Tick Time and Connection time-out combination when an intermediate node attempts to decrement this value before passing the Unconnected\_Send, Forward\_Open, or Forward\_Close service on to the next node in the path.

This shall also be caused when the Unconnected\_Send is too large to be sent out on a network connection.

**3-5.6.2.21 Message too large for unconnected message service (extended status = 0x0206)**

The message to be sent via the unconnected message service was larger than fits in the buffers allocated for the unconnected send service. This extended status code can only be detected by the originating node.

**3-5.6.2.22 Unconnected acknowledge without reply (extended status = 0x0207)**

The message to be sent via the unconnected message service was acknowledged but not responded to.

**3-5.6.2.23 No buffer memory available (extended status = 0x0301)**

The extended status code shall occur when insufficient memory is available to allocate a connection buffer. Routers and target nodes shall detect this error.

**3-5.6.2.24 Bandwidth not available (extended status = 0x302)**

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.

**3-5.6.2.25 No screeners available (extended status = 0x0303)**

Any device in the path that is a consumer and does not have a screener available shall detect this extended status code.

**3-5.6.2.26 Not configured to send real-time data (extended status = 0x304)**

If requested to make a scheduled connection, any device that is unable to send scheduled packets shall return this extended status code. For example, this code shall be returned by a node whose MAC ID is greater than SMAX.

**3-5.6.2.27 Port not available (extended status = 0x0311)**

This extended status code is the result of specifying a non-existent port in a port segment.

**3-5.6.2.28 Link address not available (extended status = 0x0312)**

This extended status code is the result of a port segment that specifies an invalid link address. This extended status code shall not be used for valid link addresses that do not respond.

**3-5.6.2.29 Invalid segment type in path (extended status = 0x0315)**

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.

**3-5.6.2.30 Error in close path (extended status = 0x0316)**

The path in the 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.

**3-5.6.2.31 Scheduling not specified (extended status = 0x0317)**

Either a Schedule Segment was expected in the path and not found, or the Encoded Value in the Schedule Segment was zero (i.e. invalid).

**3-5.6.2.32 Link address to self invalid (extended status = 0x0318)**

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.

**3-5.6.2.33 Secondary resources unavailable (extended status = 0x0319)**

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.

**3-5.6.2.34 Connection already established (extended status = 0x031A)**

A request for a direct connection has been refused because the corresponding data is already included in a larger block of data being sent via another connection.

**3-5.6.2.35 Direct connection already established (extended status = 0x031B)**

A request for a connection has been refused because a member of the corresponding data is already being sent via a direct connection.

**3-5.6.2.36 Miscellaneous (extended status = 0x031C)**

Essentially, if no other extended status code applies, then return this one.

**3-5.6.2.37 Redundant connection mismatch (extended status = 0x031D)**

This extended status code shall be returned when establishing a redundant owner connection without matching the following fields in the Forward\_Open to the same fields in the other redundant owner connections to the same target path:

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

**3-5.6.2.38 No more consumer resources available in the producing module (extended status = 0x031E)**

A node that was configured to consume data from a producing node can display this error if the producing node was not configured to support enough consumers.

**3-5.6.2.39 No connection resources exist for target path (extended status = 0x031F)**

A node that was configured to consume data from a producing node can display this error if the data on the producing node was not configured to be produced.

**3-5.6.3 General status code 0x02**

This general status code shall be returned when the Connection Manager object lacks the resources to handle a connection request.

**3-5.6.4 General status code 0x03**

This general status code shall be returned by the Get\_Connection\_Data request when the connection number supplied with the service is invalid (i.e. no connection is present). This general status code is also returned by the Search\_Connection\_Data service when the connection specified by the Connection Serial Number, Vendor Id, and Originator Serial Number is not found at the target.

**3-5.6.5 General status code 0x04**

This general status code shall be returned when there is a problem with the Segment Type in the path of a service request. The extended status provides a zero based 16-bit word offset to the place where the error occurred. The offset begins with the first word after the path size in the message.

**3-5.6.6 General status code 0x05**

This general status code shall be returned when there is a problem with the Destination in the path of a service request. The extended status provides a zero based 16-bit word offset to the place where the error occurred. The offset begins with the first word after the path size in the message.

**3-5.6.7 General status code 0x07**

The general status code shall be returned by the following services if the requests were made via a message router connection and the connection had been closed:

- Get\_Attribute\_All
- Set\_Attribute\_All
- Get\_Attribute\_Single
- Set\_Attribute\_Single
- Get\_Attribute\_List
- Set\_Attribute\_List

**3-5.6.8 General status code 0x08**

This general status code shall be returned when the connection manager object on a particular device does not support the requested service.

**3-5.6.9 General status code 0x09**

This general status code shall be returned when there is an error in the data segment. The extended status shall provide an indication of the problem. When this general status code is returned in response to a find or Set\_Attributes\_All request, the extended status shall indicate the attribute number of the first attribute that caused the error.

**3-5.6.10 General status code 0x0C**

This general status code shall be returned when the state of the target application prevents the service request from being handled. The first word of Extended Status can report the object's current state. The extended status is considered optional and is not required.

For example, an object 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, as in 3-5.6.11 (General status code 0x10).

**3-5.6.11 General status code 0x10**

This general status code shall be returned when the state of the device prevents the service request from being handled. The first word of Extended Status can report the device's current state. For the Connection Manager object, the extended status is considered optional and is not required.

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.

**3-5.6.12 General status code 0x11**

This general status code shall be returned by services to indicate that the response buffer is not large enough to hold the data, which was requested by the service. This applies in particular to Get\_Connection\_Data, Search\_Connection\_Data, Get\_Attribute\_All, Get\_Attribute\_List, and Get\_Attribute\_Single services.

**3-5.6.13 General status code 0x13**

This general status code shall be returned when not enough data was provided to perform a service request. This could mean that there wasn't enough data in the Connection Path or the size of the message was too small to even route the message to the appropriate object.

**3-5.6.14 General status code 0x14**

This general status code shall be returned by the find service when an attribute included in the service is not defined for the Connection Manager. The extended status shall indicate the attribute number of the first attribute that caused the error.

**3-5.6.15 General status code 0x15**

This general status code shall be returned when too much data was provided to perform a service request. This could mean that there was too much data in the Connection Path or the size of the message was greater than expected for a given service request.

**3-5.6.16 General status code 0x25****3-5.6.16.1 Usage**

This general status code shall be returned if the electronic key logical segment is included as part of the path and a problem was detected with the data contained in the electronic key logical segment. The extended status shall provide details of the error. This general status code shall not be returned if the electronic key logical segment was part of the connection path. In that case, general status 0x01 (Connection Failure) would be returned. This section provides details of the specific extended status codes that are to be returned.

**3-5.6.16.2 Vendor Id or Product Code error 0x0114**

The Product Code or Vendor Id specified in the electronic key logical segment does not match the Product Code or Vendor Id in the target device.

**3-5.6.16.3 Product Type error 0x0115**

The Product Type specified in the electronic key logical segment does not match the Product Type in the target device.

**3-5.6.16.4 Revision mismatch 0x0116**

The major and minor revision information in conjunction with the exact match bit specified in the electronic key logical segment does not correspond to a valid revision for this device.

**3-5.6.17 General status code 0x26**

This general status code shall be returned when the path size is invalid. This could mean that not enough or too much data is contained in the path to correctly specify the destination of a service request.

## 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 behaviour concerning the relationship between different connections each sharing a producer, and shall be one of LISTEN\_ONLY, INPUT\_ONLY, EXCLUSIVE\_OWNER, or REDUNDANT\_OWNER.

### 3-6.1 Listen only

If a connection has an application type of listen only, it shall be dependent on another connection for its existence. Devices that wish to listen to multicast data without providing configuration or scheduling information may use this application type. If the connection on which a listen only connection depends is closed or times out, the listen only connection shall also be closed.

### 3-6.2 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 scheduled input only connection, the Forward\_Open path shall contain a schedule segment. No O=>T data shall be sent; therefore, a target may accept many input only connections. A specific implementation may limit the number of input only connections it accepts.

### 3-6.3 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 data that controls outputs may be present; therefore, a target may only accept one exclusive owner connection. 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.

NOTE: Exclusive owner connections are the most common application type for controller to I/O connections since they allow the controller to receive inputs and control outputs on the same connection.

## 3-6.4 Redundant owner

### 3-6.4.1 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.

**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.

### 3-6.4.2 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.O2T\_connection\_parameters specifies whether the connection is an exclusive owner or redundant owner connection

### 3-6.4.3 Redundant owner O⇒T data format

#### 3-6.4.3.1 General

Redundant owner shall have a 32-bit header prefixed to the real-time data.

### 3-6.4.3.2 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 behaviour if more than one other connection has its COO flag set.

### 3-6.4.3.3 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 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 real-time packet, that originator application shall have the owning connection;
- If there are multiple originator applications which had the COO flag set in its last real-time 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.

### 3-6.4.5 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 real-time 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.

### 3-6.5 RUN/IDLE Notification

Applications which require RUN/IDLE notification via O=>T and/or T=>O network connections shall use at least one of the following real-time transfer formats:

- 32-bit header, with fixed or variable size network connection;
- no header, with variable size network connection.

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

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 1-4) are defined in 3-7, Redundant Owner. The reserved field (bits 4-31) shall be reserved and set to 0.

If the no header transfer format is used, the reception of a packet with data beyond the transport header shall indicate the RUN mode. If the packet is truncated after the transport header, the target application shall be sent an idle event. The real-time transfer formats described in this section shall only apply to transport types LISTEN\_ONLY, INPUT\_ONLY and EXCLUSIVE\_OWNER. The real-time transfer formats for the REDUNDANT\_OWNER transport type is described in 3.5.

**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 Connection Manager 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	All Ports	ARRAY of STRUCT UINT UINT	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.	

**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
01 <sub>hex</sub>	Optional	Get_Attribute_All	Returns the contents of all attributes of the class.
0E <sub>hex</sub>	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 Implementation	Access Rule	Attribute Name	Data Type	Description of Attribute	Semantics
1	Required	Set	Port Type	UINT		0 = connection terminates in this device 1 = reserved for compatibility with existing protocols 2 = ControlNet 3 = ControlNet redundant 4 = TCP/IP 5 = DeviceNet 6 – 99 = reserved for compatibility with existing protocols 100 – 199 = Vendor Specific 200 – 65534 = Reserved for future use 65535 = unconfigured port
2	Required	Get	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).
3	Required	Get	Port Object	UINT	Number of 16 bit words in the following path	Range = 2 - 6
				Padded EPATH	Logical path segments that identify the object which maintains the port. For example, this could be the TCP/IP Interface Object.	The path is restricted to one logical class segment and one logical instance segment. The maximum size is 12 bytes. See Appendix C, Logical Segments.
4	Required	Get	Port Name	SHORT_STRING	String which names the port. The maximum number of characters in the string is 64. For example, this may be "Port A".	

Attr ID	Need In Implementation	Access Rule	Attribute Name	Data Type	Description of Attribute	Semantics
5	Optional	Get	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	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	Node Address <sup>2</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	Port Node Range <sup>1</sup>	UINT	Minimum node number on port.	
				UINT	Maximum node number on port.	
9	Optional	Get	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.	

<sup>1</sup> If a device can report its port characteristics within the range allowed (e.g. DeviceNet MACID) then it shall not support this attribute. Otherwise (e.g. EtherNet/IP IP Address) it shall not support this attribute.

<sup>2</sup> 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).

### 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
01 <sub>hex</sub>	Optional	Get_Attribute_All	Returns the contents of all attributes of the class.
05 <sub>hex</sub>	Optional	Reset	
0E <sub>hex</sub>	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.
10 <sub>hex</sub>	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: CIP Common Specification**

### **Chapter 4: How to Read Specifications in the Object Library**

---

This page is intentionally left blank

## 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.

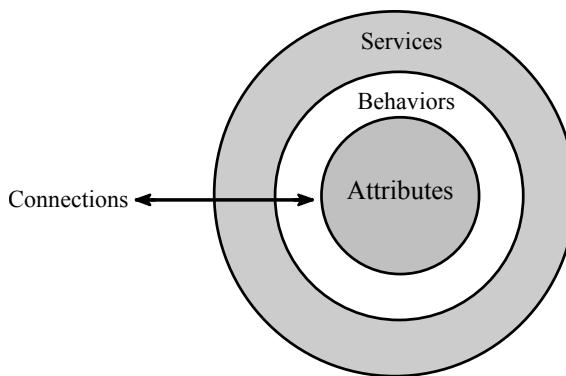
For information about	Go to Section
Reading an Object Specification	4-1
Description of Object	4-2
Class Code	4-3
Attributes	4-4
Common Services	4-5
Object-specific Services	4-6
Behavior	4-7
Connections	4-8

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:

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

### 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:

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

**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.

**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 or a special case processing method for the Client to process the error message that will occur when accessing those objects that choose not to implement the class attribute. The “Attribute Not Supported” (0x14) is the required error code.

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.

**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.

Name refers to the attribute.

**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.

Description of Attribute provides general information about the attribute.

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 4-4.1. Reserved Class Attributes for All Object Class Definitions**

<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>	<b>Semantics of Values</b>
1	Conditional	Get	<b>Revision</b>	UINT	Revision of this object Note: All class definitions are required to include this class attribute.	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	<b>Max Instance</b>	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	<b>Number of Instances</b>	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	<b>Optional attribute list</b>	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.

**Table 4-1.1. Reserved Class Attributes for All Object Class Definitions, continued**

Attr 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.

Attribute ID	Need in Implementation	Access Rule	NV	Name	DeviceNet 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 Appendix A of Volume I.

The Common Service component of an object definition includes:

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 three specifications:

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

**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*.

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 Appendix A of Volume 1 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 must be included in the object definition. This component specifies the sequence or order of the data returned in the Service Data portion of Explicit Response Message. The following byte array is an example of how the format of the Service Data portion of a Get\_Attributes\_All response is typically specified:

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)							

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
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.

**Important:** If the instance attribute "Number of Attributes" is not supported, the default value of zero is to be inserted into the response byte array and **no** attribute ID numbers will follow.

**Important:** If the instance attribute "Optional Attribute List" is not supported, the default value of zero is to be inserted for the "number of attributes" and **no** attribute ID numbers will follow.

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

- Default values must 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 an existing object, those attributes must be added to the end of the response byte array.

## 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 component specifies the sequence or order of the data supplied in the Service Data portion of the request. 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:

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)							

**Important:** It is important to note that 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 as settable.

## 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 the DeviceNet Standalone Specification).

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

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 three specifications:
  - Optional; or
  - Required; or
  - Not applicable (“n/a”)

**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*.

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.

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

The second table is the description of the service response data and includes:

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 Volume 1, 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.

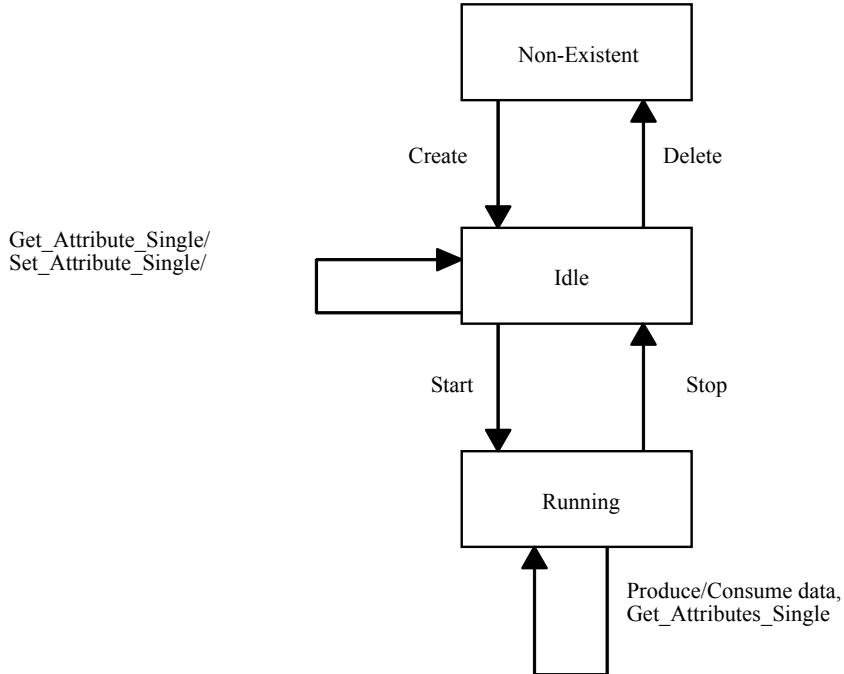
For example, the table below shows three states (other objects may or may not have other states):

- **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.

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.



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.

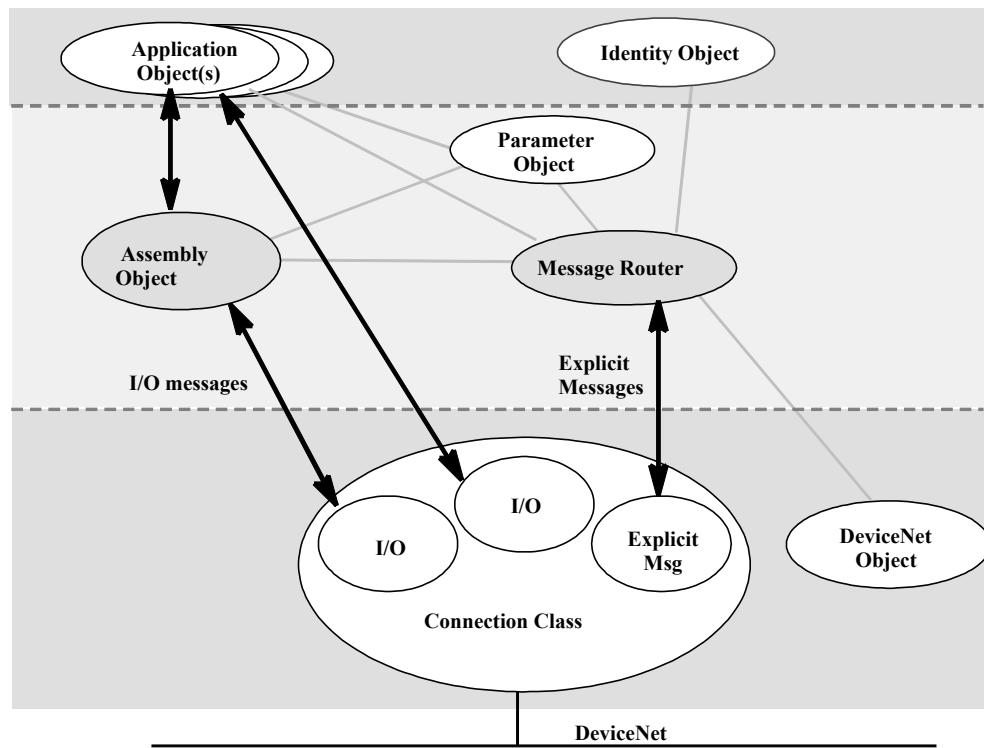
Attribute	State		
	Non-existent	Idle	Running
Number_of_Members	Not available	Read Only	Read Only
Member_List	Not available	Read Only	Read Only
Data	Not available	Read/Write	Read Only
Owner	Not available	Read Only	Read Only

## 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

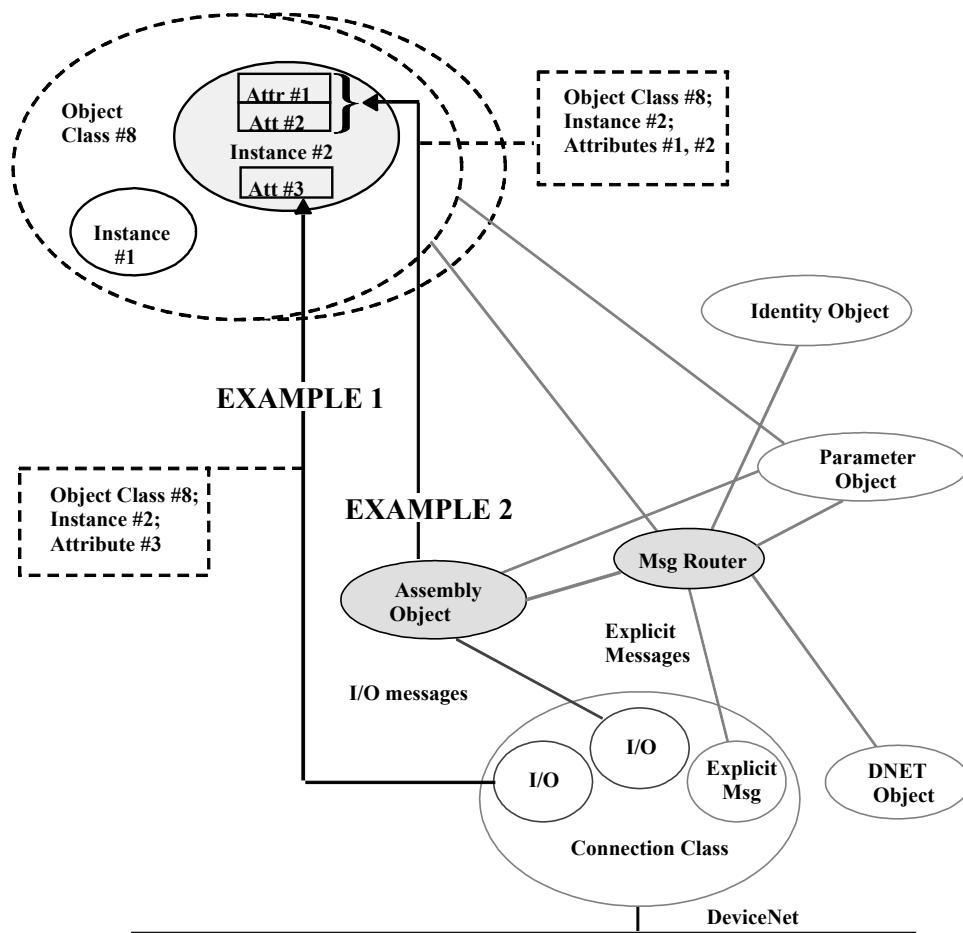
## 4-8.1 Access Through Explicit Messaging Connections

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 Chapter 5 of Volume 1, section 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.

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.

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:

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 4-9.1. defines the ranges applicable to Class ID values.

**Table 4-9.1. 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> - FF <sub>hex</sub>	Reserved by CIP for future use	56
100 <sub>hex</sub> - 2FF <sub>hex</sub>	Open	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

Table 4-9.2. defines the ranges applicable to Attribute ID values.

**Table 4-9.2. 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

Table 4-9.3. defines the ranges applicable to Service Code values.

**Table 4-9.3. 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.

An extension of an existing object definition:

- can add new attributes to the most recent definition. Deletion of attributes is not allowed.
- can only add to the **end** of the Get\_Attributes\_All and the Set\_Attributes\_All structure definitions.
- can add to the list of services. Deletions of services is not allowed.

You can make two primary types of extensions to open objects depending on your preference (when making a vendor-specific extension, you can choose to publish or not):

- vendor-specific
  - published
  - unpublished
- open

### 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.

#### 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	

#### 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 Generic Object	

This object supports these Common Services:

- Get\_Attributes\_All
- Set\_Attributes\_All

These services specify the following attributes:

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:

- Add an attribute(s):** Because you can add new attributes only after the last attribute in the most recent definition, the Instance Attribute *Input Value* is assigned a value of 100. The new list of Instance Attributes looks like this:

**Table 4-9.4. 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 Generic Object	
<b>100</b>	<b>Set</b>	<b>Input Value</b>	<b>BOOL</b>		<b>0 = off; 1 = on</b>

- Define new service structures to accommodate this new attribute:** In this example, the access rule for the new attribute is specified as Get and Set. As a result, you need to specify the new structures of both services. Because you can add only to the **end** of the Get\_Attributes\_All and Set\_Attributes\_All structure definitions, the new service structures are:

Service	Attributes
Get_Attributes_All	1, 2, 3, 100
Set_Attributes_All	2, 100

- Add to list of object specific services, if necessary.**
- 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\_All\_Attributes service response
- Set\_All\_Attributes 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.

**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.

#### 4–9.2.1.2.2 Get\_Attributes\_All Service Response

When a Client is interfacing with an object, it must be prepared to accept or ignore data supplied for the vendor-specific extension (in addition to the specified open attributes) in a Get\_Attributes\_All response.

#### 4–9.2.1.2.3 Set\_Attributes\_All Service Response

Processing a Set\_All request for a vendor-specific object or attribute may require processing additional vendor-specific data than is expected by a Server. As a result, the Server object may process a Set\_All request that lacks necessary data because the vendor-specific attributes were not recognized.

The unrecognized attributes are either *critical* or *non-critical* to the behavior of the object.

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 (00 - 63<sub>hex</sub>) 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 4-9.5.

**Table 4-9.5. 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
AttributeID – Vendor-Specific	64 <sub>hex</sub> - C7 <sub>hex</sub>	100
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:

Advantages	Disadvantage
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 4.6.

**Table 4-9.6. Open Address Ranges**

Type	Range	Quantity
Class ID	00 - 63 <sub>hex</sub>	100
	100 <sub>hex</sub> - 2FF <sub>hex</sub>	512
AttributeID	00 - 63 <sub>hex</sub>	100
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.

## **Volume 1: CIP Common Specification**

### **Chapter 5: Object Library**

---

This page is intentionally left blank

## 5-1. OBJECT SPECIFICATIONS

Using the format previously defined, the objects listed in Table 5.1. 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. Object Specifications in the CIP Object Library**

Class Code	For information about this Object:	Go to page:
01 <sub>hex</sub>	Identity	5-6
02 <sub>hex</sub>	Message Router	5-17
03 <sub>hex</sub>	DeviceNet	See the ODVA DeviceNet Specification
04 <sub>hex</sub>	Assembly	5-21
05 <sub>hex</sub>	Connection	5-28
06 <sub>hex</sub>	Connection Manager	5-29
07 <sub>hex</sub>	Register	5-30
08 <sub>hex</sub>	Discrete Input Point	5-32
09 <sub>hex</sub>	Discrete Output Point	5-39
0A <sub>hex</sub>	Analog Input Point	5-48
0B <sub>hex</sub>	Analog Output Point	5-56
0E <sub>hex</sub>	Presence Sensing	5-67
0F <sub>hex</sub>	Parameter	5-71
10 <sub>hex</sub>	Parameter Group	5-83
12 <sub>hex</sub>	Group	5-87
1D <sub>hex</sub>	Discrete Input Group	5-92
1E <sub>hex</sub>	Discrete Output Group	5-97
1F <sub>hex</sub>	Discrete Group	5-103
20 <sub>hex</sub>	Analog Input Group	5-107
21 <sub>hex</sub>	Analog Output Group	5-112

<b>Class Code</b>	<b>For information about this Object:</b>	<b>Go to page:</b>
22 <sub>hex</sub>	Analog Group	5-118
23 <sub>hex</sub>	Position Sensor Object	5-123
24 <sub>hex</sub>	Position Controller Supervisor Object	5-129
25 <sub>hex</sub>	Position Controller Object	5-136
26 <sub>hex</sub>	Block Sequencer Object	5-146
27 <sub>hex</sub>	Command Block Object	5-148
28 <sub>hex</sub>	Motor Data Object	5-154
29 <sub>hex</sub>	Control Supervisor Object	5-159
2A <sub>hex</sub>	AC/DC Drive Object	5-167
2B <sub>hex</sub>	Acknowledge Handler Object	5-176
2C <sub>hex</sub>	Overload Object	5-186
2D <sub>hex</sub>	Softstart Object	5-189
2E <sub>hex</sub>	Selection Object	5-195
30 <sub>hex</sub>	S-Device Supervisor Object	5-205
31 <sub>hex</sub>	S-Analog Sensor Object	5-219
32 <sub>hex</sub>	S-Analog Actuator Object	5-249
33 <sub>hex</sub>	S-Single Stage Controller Object	5-256
34 <sub>hex</sub>	S-Gas Calibration Object	5-263
35 <sub>hex</sub>	Trip Point Object	5-269
F0 <sub>hex</sub>	ControlNet Object	See the ControlNet International Specification
F1 <sub>hex</sub>	ControlNet Keeper Object	See the ControlNet International Specification
F2 <sub>hex</sub>	ControlNet Scheduling Object	See the ControlNet International Specification
F3 <sub>hex</sub>	Connection Configuration Object	5-276
F4 <sub>hex</sub>	Port Object	3-84

Class Code	For information about this Object:	Go to page:
F5 <sub>hex</sub>	TCP/IP Interface Object	See the EtherNet/IP Specification, Volume 2
F6 <sub>hex</sub>	EtherNet Link Object	See the EtherNet/IP Specification, Volume 2

## 5-2. IDENTITY OBJECT

Class Code: 01hex

This object provides identification of and general information about the device. The Identity Object MUST 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

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.					

<sup>1</sup> 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

Attr ID	Need in implementation	Access Rule	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

Attr ID	Need in implementation	Access Rule	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 Attribute 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.

The following Instance Attributes are used to identify with certainty the appropriate device targeted by a connection originator:

- Vendor
- 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”.

### Semantics:

#### *Vendor ID*

Vendor IDs are managed by the Open DeviceNet Vendor Association, Inc. (ODVA) and ControlNet International (CI). The value zero is not valid.

#### *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 on page 5-3.

#### *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.

### **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 effect the configuration choices available to the user (and therefor the Electronic Data Sheet) require incrementing the major revision.

The minor revision is typically used to identify changes in a product that do not effect 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.

### **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 5-2.1. Bit Definitions for Status Instance Attribute of Identity Object**

Bit (s)	Called	Definition
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.
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.

Bit (s)	Called	Definition
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 5-2.2. Bit Definitions for Extended Device Status Field**

Bits 4 - 7:	Extended Device Status Description
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:

- 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.

	Recoverable	Non-recoverable
<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 5-2.3.

#### **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.

**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.

**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.

**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.

**Heartbeat Interval:**

This attribute sets the nominal interval between production of optional heartbeat messages.

### 5-2.3. Common Services

The Identity Object provides the following Common Services:

Service Code	Need in Implementation		Service Name	Description of Service
	Class	Instance		
0E <sub>hex</sub>	Conditional <sup>1</sup>	Conditional <sup>2</sup>	Get_Attribute_Single	Returns the contents of the specified attribute.
05 <sub>hex</sub>	Optional	Required	Reset	Invokes the Reset service for the device.
01 <sub>hex</sub>	Optional	Conditional <sup>2</sup>	Get_Attributes_All	Returns a predefined listing of this objects attributes (See the Get_Attributes_All Response definition below)
10 <sub>hex</sub>	n/a	Conditional	Set_Attribute_Single	Modifies an attribute (Required if Heartbeat interval is defined)
11 <sub>hex</sub>	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.

<sup>1</sup>The Get\_Attribute\_Single service is REQUIRED if any Class attributes are implemented.

<sup>2</sup>Either the Get\_Attribute\_Single or the Get\_Attributes\_All service shall be supported at a minimum.

#### 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:

Name	Type	Description of Request Parameters	Semantics of Values
Type	USINT	Type of Reset	See Table below.

The parameter Type for the Reset common service has the following bit specifications:

Value:	Type of Reset:
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 out-of-box configuration, then emulate cycling power as closely as possible.

### 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:

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.

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:

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
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)							
11	Serial Number							
12	Serial Number							
13	Serial Number (high byte)							
14	Product Name length							

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
15	Product Name (1st character)							
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							

**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.

## 5-2.4. Object-specific Services

The Identity Object provides no Object-specific services.

## 5-2.5. Behavior

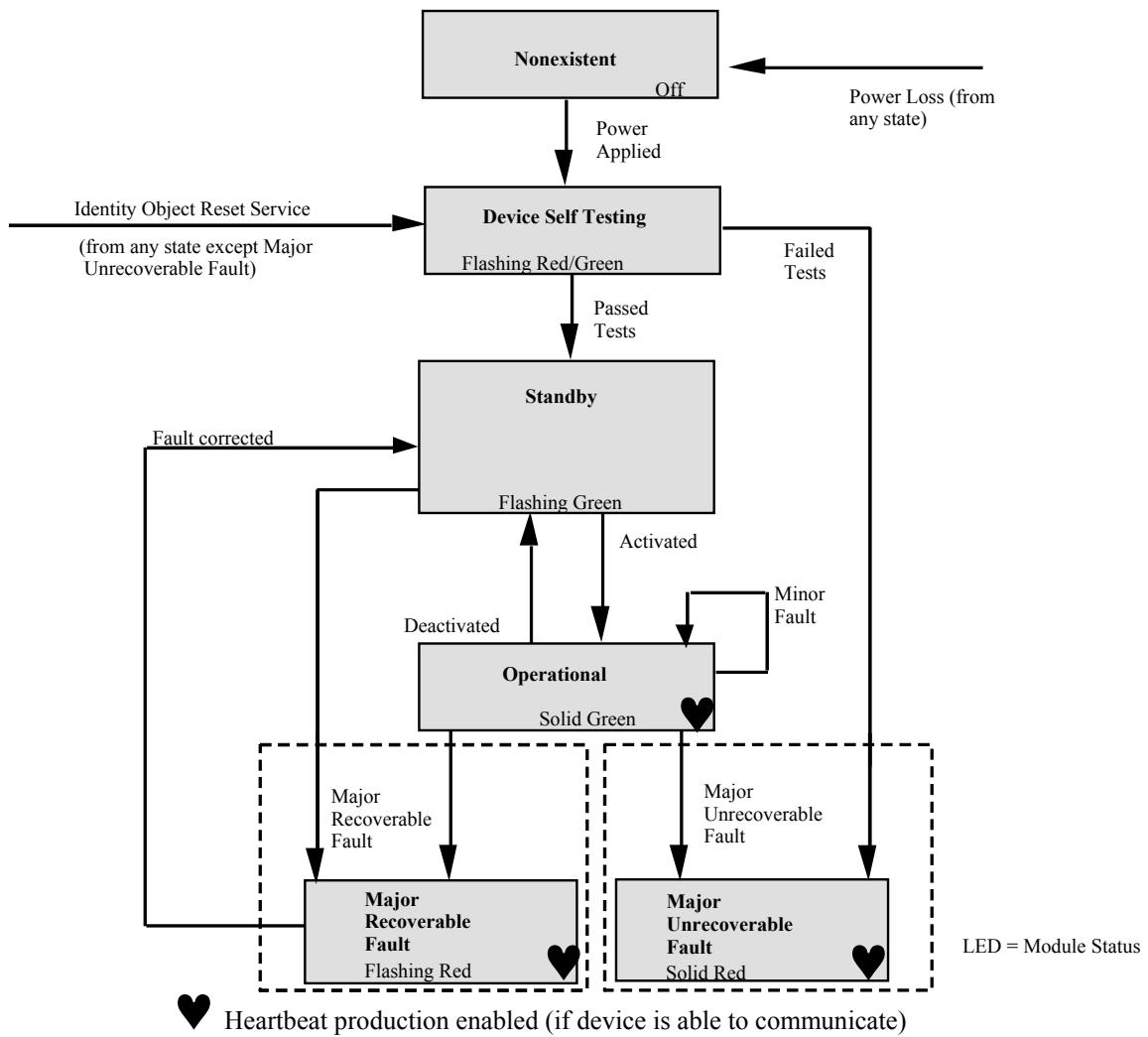
The behavior of the Identity Object is illustrated in the State Transition Diagram (STD) in Figure 52.3. This STD associates the state of the device with the status reported by the Status Attribute with the state of the Module Status LED.

**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.

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 5-2.3.** 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

**Table 5.4. 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

This page is intentionally left blank

## 5-3. MESSAGE ROUTER OBJECT

Class Code: 02hex

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

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

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:

Service Code	Need in Implementation		Service Name	Description of Service
	Class	Instance		
0E <sub>hex</sub>	Conditional*	Conditional*	Get_Attribute_Single	Returns the contents of the specified attribute.
01 <sub>hex</sub>	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:

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 (see Chapter 4 of Volume I for a description of the Service Data field) of the Get\_Attributes\_All response is as follows:

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
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

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 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.

**Important:** Insert default values in place of unsupported attributes.

### 5-3.4. Object-Specific Services

The Message Router Object provides no Object-specific services.

### 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

#### 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.

#### 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: 03hex

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 DeviceNet standalone specification for the definition of this object class.

## 5-5. ASSEMBLY OBJECT

Class Code: 04hex

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.

### Revision History

Connection 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 5-5.1. 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> - FF <sub>hex</sub>	Reserved by CIP for future use	56
100 <sub>hex</sub> - 2FF <sub>hex</sub>	Open (static assemblies defined in device profile)	512
300 <sub>hex</sub> - 4FF <sub>hex</sub>	Vendor Specific static assemblies and dynamic assemblies	512
500 <sub>hex</sub> - FFFF <sub>hex</sub>	Reserved by CIP for future use	64,256

### 5-5.1. 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

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		

### 5-5.3. Common Services

The Assembly Object provides the following Common Services:

Service Code	Need in Implementation				Service Name	Description of Service		
	Static Assembly		Dynamic Assembly					
	Class	Instance	Class	Instance				
0E <sub>hex</sub>	Conditional <sup>1</sup>	Required	Required	Required	Get_Attribute_Single	Returns the contents of the specified attribute.		
08 <sub>hex</sub>	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.		
10 <sub>hex</sub>	n/a	Optional	n/a	Conditional <sup>2</sup>	Set_Attribute_Single	Modifies an attribute value.		
09 <sub>hex</sub>	n/a	n/a	Optional <sup>3</sup>	Required	Delete	Deletes an Assembly Object and releases all associated resources.		

Service Code	Need in Implementation				Service Name	Description of Service		
	Static Assembly		Dynamic Assembly					
	Class	Instance	Class	Instance				
1A <sub>hex</sub>	n/a	n/a	n/a	Conditional <sup>4</sup>	Insert_Member	Adds a member to the Assembly Member List.		
1B <sub>hex</sub>	n/a	n/a	n/a	Conditional <sup>4</sup>	Remove_Member	Removes a member from the Assembly Member List.		
18 <sub>hex</sub>	n/a	Optional	n/a	Optional	Get_Member	Returns a member from the Assembly Member List.		
19 <sub>hex</sub>	n/a	n/a	n/a	Optional	Set_Member	Modifies a member of the Assembly Member List.		

<sup>1</sup>Required if Max Instance is implemented or Instance Attribute 2 (Member List).

<sup>2</sup>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.

<sup>3</sup>At the class level this service deletes all existing Assembly instances.

<sup>4</sup>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.

## 5-5.4. Object-specific Services

The Assembly Object provides no Object-specific services. The following Object-specific service codes have been obsoleted:

Obsoleted Service Code	Need in Implementation				Service Name	Description of Service		
	Static Assembly		Dynamic Assembly					
	Class	Instance	Class	Instance				
4B <sub>hex</sub>	n/a	n/a	n/a	n/a	Add_Member	Obsolete		
4C <sub>hex</sub>	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.

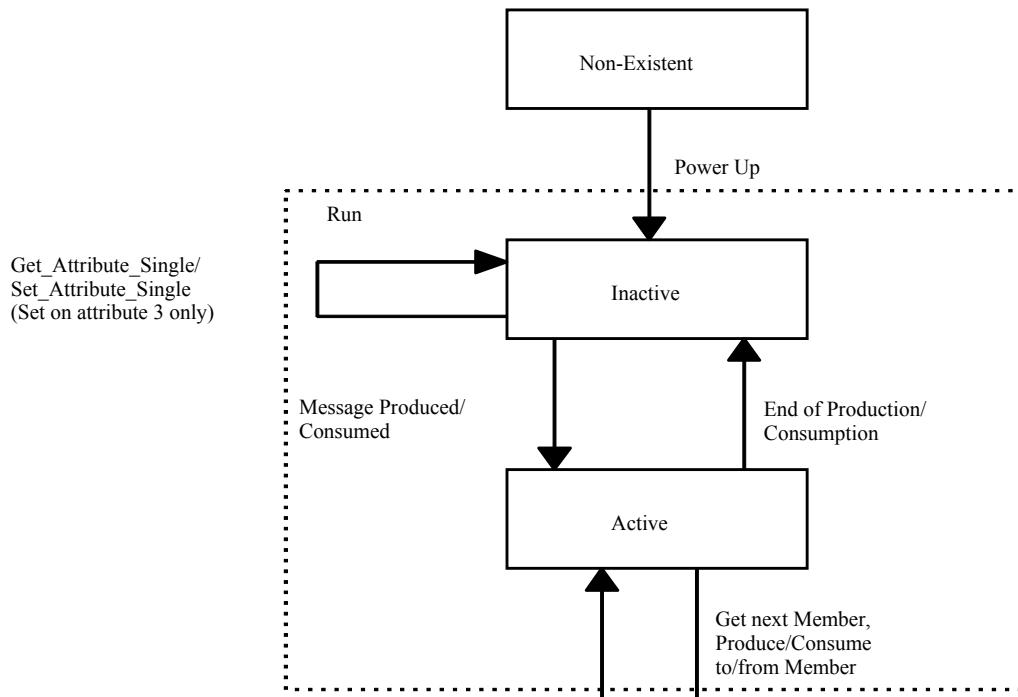
The empty path shall be supported for all dynamic assemblies.

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.



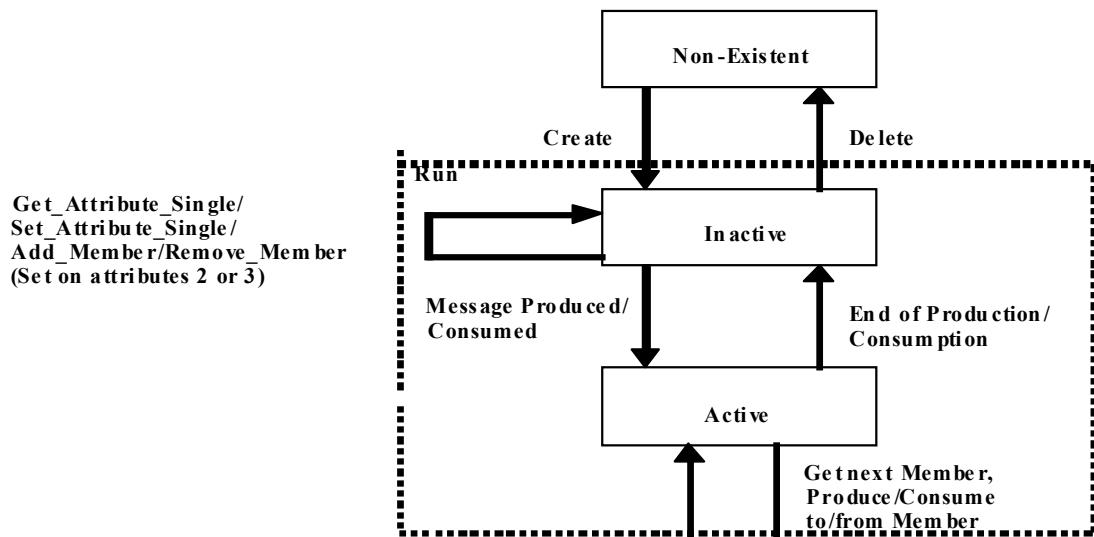
Event	Static Assembly Object State		
	Non-existent	Inactive	Active
Power Up	Transition to Inactive	Not applicable	Not applicable
Get_Attribute_Single	Error: Object does not exist. (General Error Code 16 <sub>hex</sub> )	Validate/service the request Return response	Validate/service the request Return 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 5-5.2. 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.



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 5-5.3. 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

<sup>1</sup>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: 05hex

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

Class Code: 06hex

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: 07hex

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

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

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. See the encoding example below.

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:

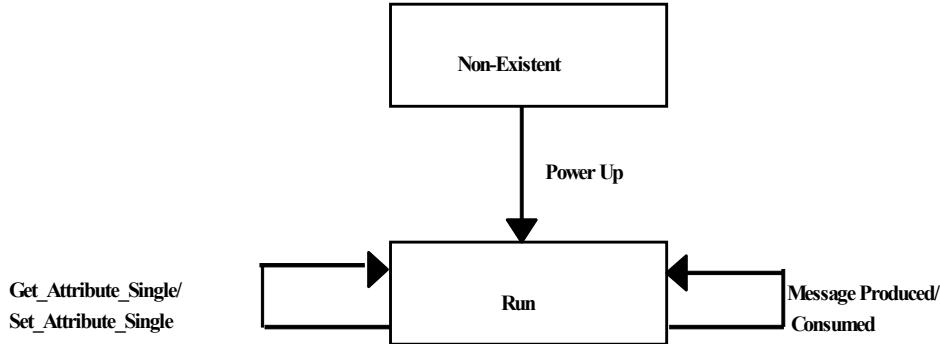
Service Code	Need in Implementation		Service Name	Description of Service
	Class	Instance		
0E <sub>hex</sub>	Conditional*	Required	Get_Attribute_Single	Returns the contents of the specified attribute.
10 <sub>hex</sub>	n/a	Optional	Set_Attribute_Single	Modifies an attribute value.

\*The Get\_Attribute\_Single service is REQUIRED if any attributes are implemented.

### 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.

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: 08hex

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:

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

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

Attribute ID	Need in implementation	Access Rule	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.

<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.

### 5-9.4. Common Services

The Discrete Input Point Object provides the following Common Services:

Service Code	Need in Implementation		Service Name	Description of Service
	Class	Instance		
0E <sub>hex</sub>	Required	Required	Get_Attribute_Single	Returns the contents of the specified attribute.
01 <sub>hex</sub>	Optional	Optional	Get_Attributes_All	Returns a predefined listing of this objects attributes (See the Get_Attributes_All Response definition below)
10 <sub>hex</sub>	n/a	Optional	Set_Attribute_Single	Modifies an attribute value.
02 <sub>hex</sub>	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 (see Chapter 4 of Volume I for a description of the Service Data field) of the Get\_Attributes\_All response is as follows:

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:

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:

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

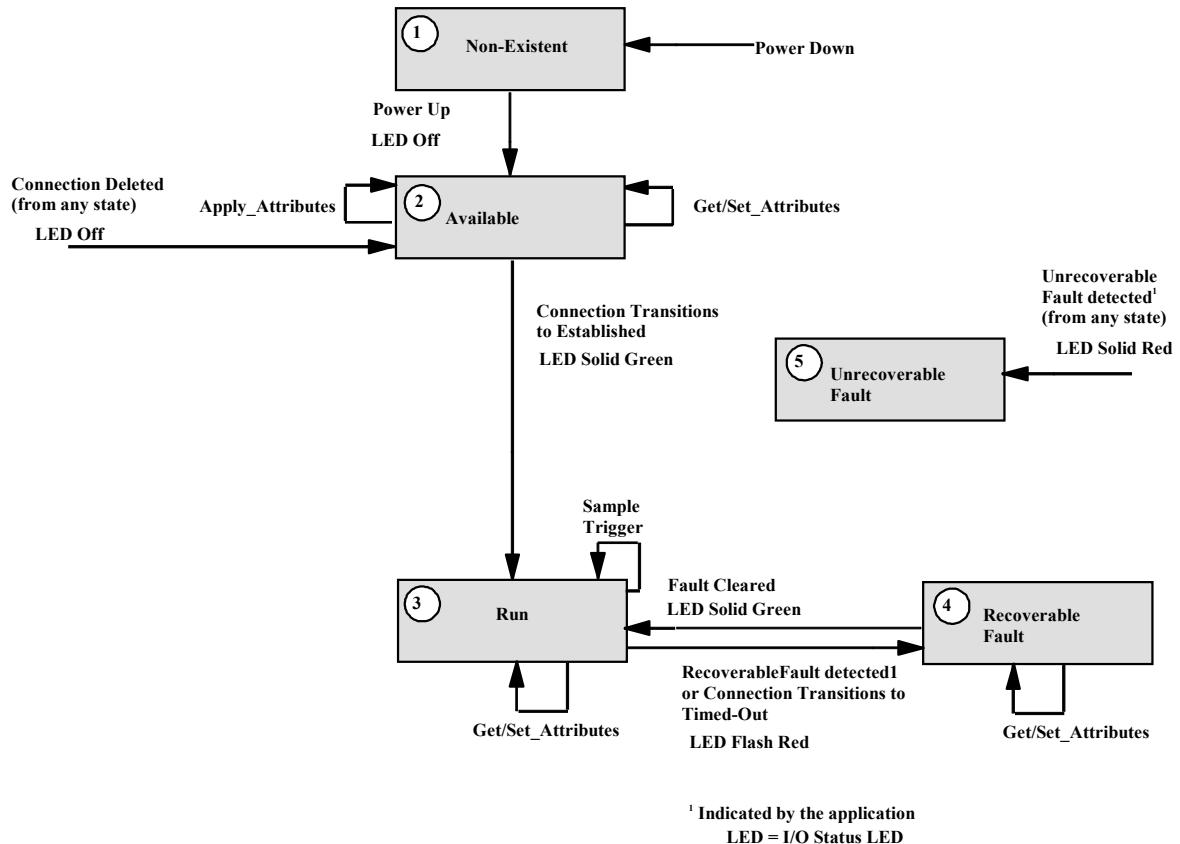
The DIP Object provides no Object-specific services.

### 5-9.6. Behavior

The State Transition Diagram in Figure 5-9.1. 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 5-9.2. 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 5-9.1.** 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:

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

This event	Is
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.”

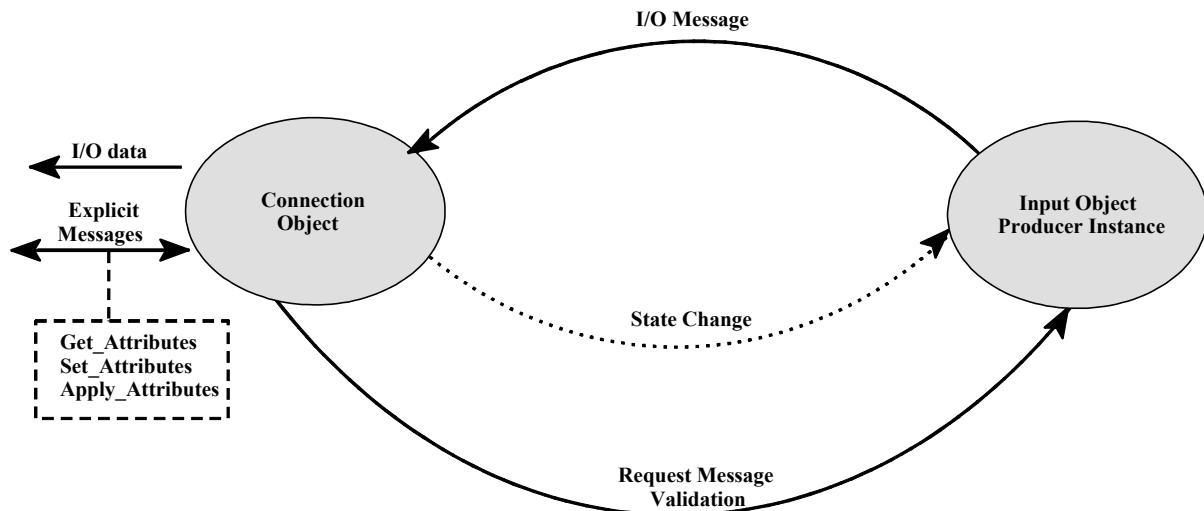


Table 5-9.2. State Event Matrix for the Discrete Input Point Object

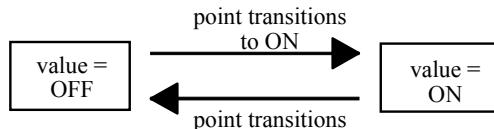
Event	State				
	Non-Existent	Available	Run	Recoverable Fault	Unrecoverable Fault
Sample Trigger	Not Applicable	Ignore event	Sample data, Send data	Ignore event	Ignore event
Apply Attributes	Not Applicable	Verify attributes, return result	Return error (Object State Conflict)	Return error (Object State 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

Event	State				
	Non-Existent	Available	Run	Recoverable Fault	Unrecoverable Fault
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

### 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.



The optional *Status* Instance attribute is simply a logical OR of all possible failure or alarm conditions for the point.



## 5-10. DISCRETE OUTPUT POINT OBJECT

Class Code: 09hex

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

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

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
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

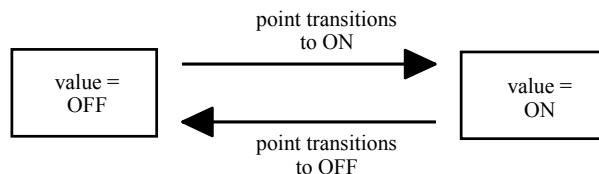
Attribute ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
10	Optional	Set	Flash	BOOL	Flash output at periodic rate if point is ON	0=no flash; 1=flash
11	Optional	Set	Flash Rate	USINT	Flash Rate for Flash attribute	unsigned positive integer indicating 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

**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:

Attribute ID	Name	Default Value
5	Fault Action	0
6	Fault Value	0
7	Idle Action	0
8	Idle Value	0
10	Flash	0

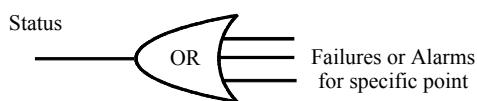
### 5-10.2.1 Value

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.



### 5-10.2.2 Status

The *Status* attribute is simply a logical OR of all possible failure or alarm conditions for the point.



### 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

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:

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.

Upon entering the Recoverable\_Fault state the DOP will behave according to the following table.

	Fault_State = 0	Fault_State = 1
<b>Fault_Value = 0</b>	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.
<b>Fault_Value = 1</b>	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:

Service Code	Need in Implementation		Service Name	Description of Service
	Class	Instance		
0E <sub>hex</sub>	Condition al*	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.
01 <sub>hex</sub>	Optional	Optional	Get_Attributes_All	Returns a predefined listing of this objects attributes (See the Get_Attributes_All Response definition below)
02 <sub>hex</sub>	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:

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:

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
.	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
.	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.

**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:

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.

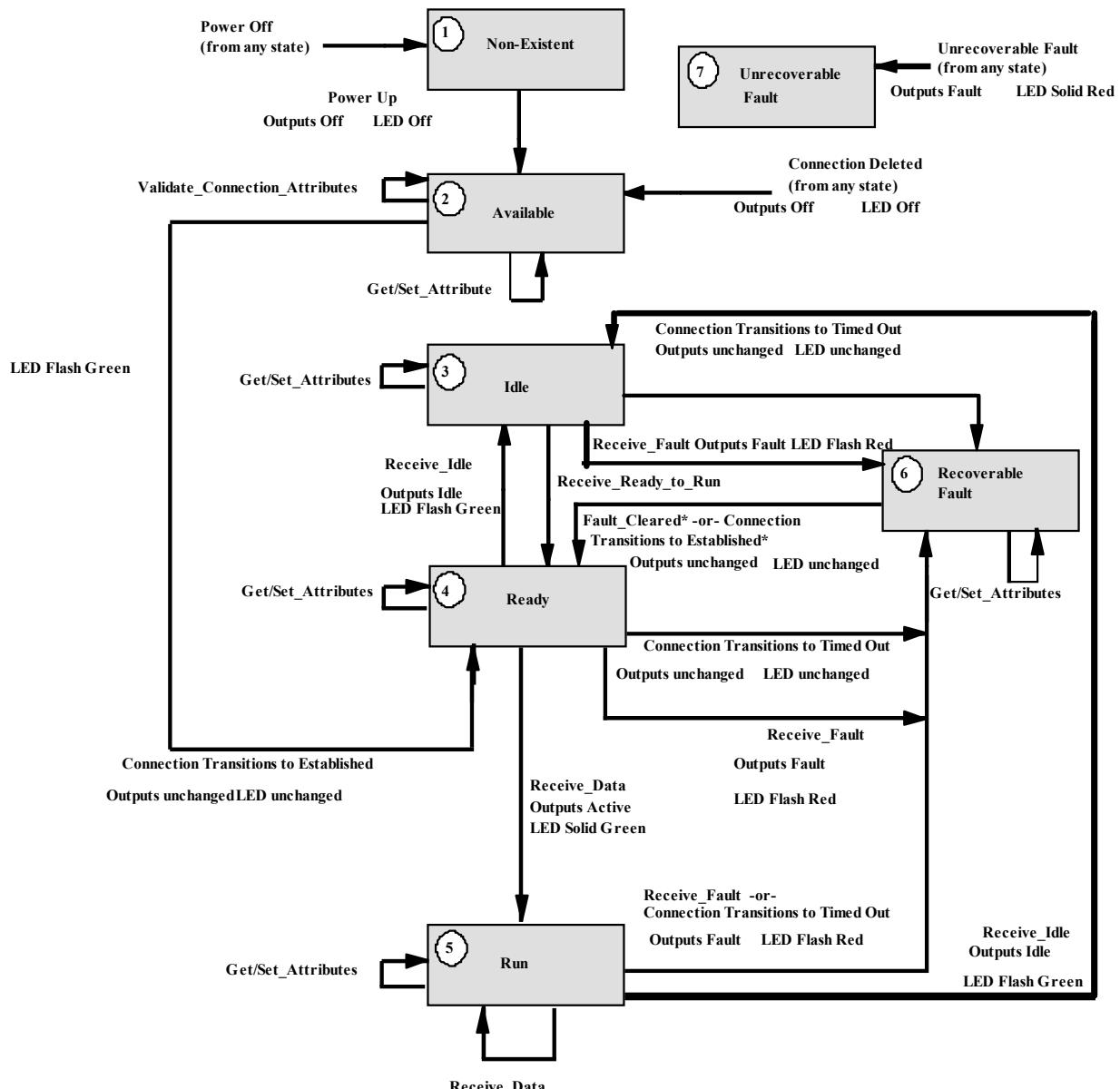
#### 5-10.5. Behavior

The State Transition Diagram in Figure 5-10.1. 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.

**Figure 5-10.1.** State Transition Diagram for Discrete Output Point Object



**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.

The SEM also contains these events:

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 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.

**Table 5-10.2 State Event Matrix for the Analog Input 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 <b>Verify and Accept Data Transition to Run</b> Ignore event	LED Solid Green, Accept Data, Transition to <b>Run</b>	If data length is zero <sup>2</sup> Transition to Idle Otherwise <b>Verify and Accept Data</b>	Ignore event	Ignore event
Receive_Fault	Not applicable	Not applicable	Transition to <b>Recoverable Fault</b> <sup>1</sup>	Transition to <b>Recoverable Fault</b> <sup>1</sup>	Transition to <b>Recoverable Fault</b> <sup>1</sup>	Ignore event	Ignore event
Receive_Idle	Not applicable	<b>Return Error (object state conflict)</b>	Ignore event	Transition to <b>Idle</b> <sup>2</sup>	Transition to <b>Idle</b> <sup>2</sup>	<b>Return Error (object state conflict)</b>	Ignore event
Receive_Ready_to_Run	Not applicable	<b>Return Error (object state conflict)</b>	Transition to <b>Ready</b>	Ignore event	Ignore event	<b>Return Error (object state conflict)</b>	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 <b>Available</b>	Transition to <b>Available</b>	Transition to <b>Available</b>	Transition to <b>Available</b>	Ignore event
Connection Transition to Established	Not applicable	Transition to <b>Ready</b>	Return Error (object state conflict)	Return Error (object state conflict)	Return Error (object state conflict)	<b>Transition to Ready</b> <sup>3</sup>	Ignore event
Connection transitions to Timed Out state	Not applicable	Not applicable	Transition to <b>Recoverable Fault</b>	Transition to <b>Recoverable Fault</b>	Transition to <b>Recoverable Fault</b> <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	<b>Return Error (Object Does Not Exist)</b>	Verify, and Accept, and Apply value	Verify, and Accept, and Apply value	<b>Verify, and Accept, and Apply value</b>	<b>Verify, and Accept, and Apply value</b>	<b>Verify, and Accept, and Apply value</b>	Ignore event
Fault_Cleared	Ignore event	Ignore event	Ignore event	Ignore event	Ignore event	Transition to <b>Ready</b> <sup>3</sup>	Ignore event

1 - Fault: Hold Last State OR use Fault Value.-

2 - Idle: Hold Last State OR use Idle Value.

**3 - If no other faults exist (Note, a Connection time out is considered a fault)**

### 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: 0Ahex

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:

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

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

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.	

Attr ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
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:

Service Code	Need in Implementation		Service Name	Description of Service
	Class	Instance		
0E <sub>hex</sub>	Required	Required	Get_Attribute_Single	Returns the contents of the specified attribute.
10 <sub>hex</sub>	n/a	Conditional <sup>1</sup>	Set_Attribute_Single	Modifies an attribute value.
01 <sub>hex</sub>	Optional	Optional	Get_Attributes_All	Returns a predefined listing of this objects attributes (See the Get_Attributes_All Response definition below)
02 <sub>hex</sub>	n/a	Optional	Set_Attributes_All	Modifies the contents of the attributes of the class or object.

<sup>1</sup>The Set\_Attribute\_Single service is required only if Instance Attributes 7 and/or 8 are implemented.

### 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:

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.

At the Instance 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:

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:

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

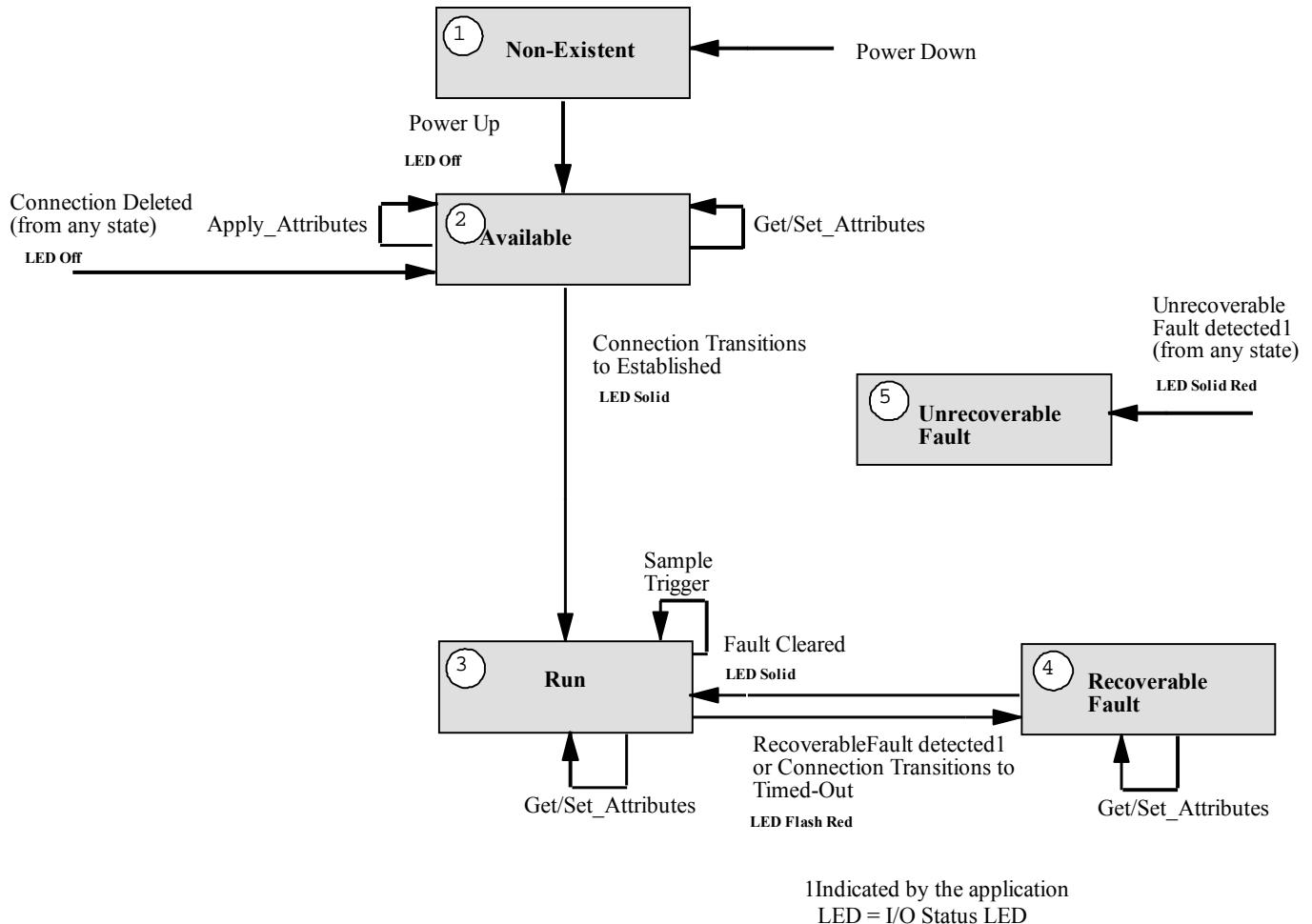
The Analog Input Object provides no Object-specific services:

#### 5-11.6. Behavior

The State Transition Diagram in Figure 5-11.1. 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. A subset of the states and events may be supported in an application, but the behavior must still be consistent.

Figure 5-11.1. State Transition Diagram for Analog Input Point Object



**Important:** Events can occur simultaneously, but the *Fault* events have priority if they occur simultaneously with other events.

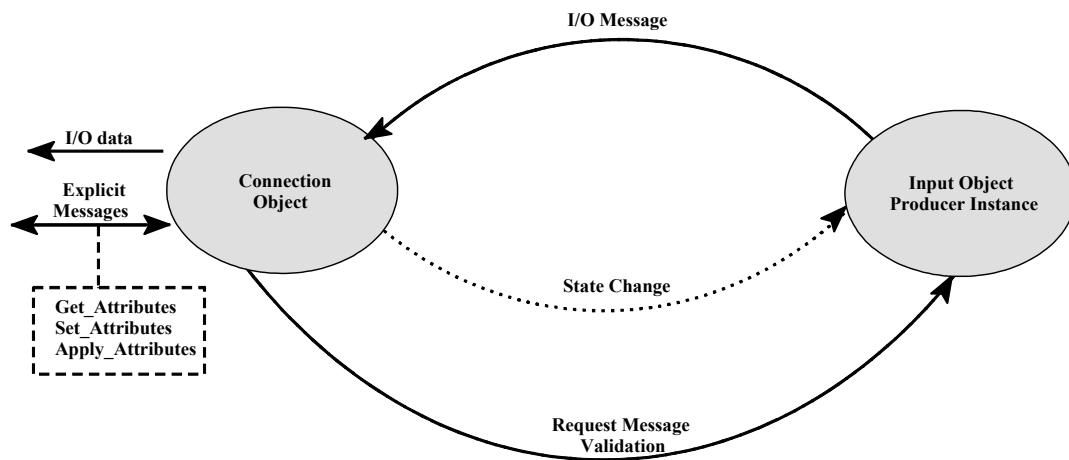
The SEM contains the following states:

- **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.

The SEM also contains these events:

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	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.”



**Table 5-11.2. 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

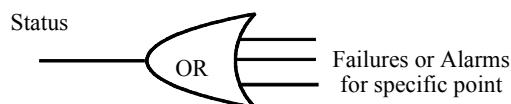
### Attribute Access Rules

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.

The optional *Status* attribute is simply a logical OR of all possible failure or alarm conditions for the point.



The *Owner Vendor ID* and *Owner Serial Number* are included in the object definition but their use is TBD.

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.

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: 0Bhex

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

Attr 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.					

### 5-12.2. 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	

Attr ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
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 1 = REAL 2 = USINT 3 = SINT 4 = DINT 5 = LINT 6 = UINT 7 = UDINT 8 = ULINT 9 = LREAL 100 = vendor specific
9	Optional	Set	Fault State	USINT	Output value to go to on failure or fault	0 = hold last state 1 = low limit 2 = high limit 3 = user specified value
10	Optional	Set	Idle State	USINT	Output value to go to on idle mode	0 = hold last state 1 = low limit 2 = high limit 3 = user specified value
11	Optional	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	Optional	Set	Command	BOOL	Changes state of AOP to Idle Mode or Run Mode	0 = idle 1 = run

Attr ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
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

**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:

Attribute ID	Name	Default Value
7	OutputRange	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:

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.
01 <sub>hex</sub>	Optional	Optional	Get_Attributes_All	Returns a predefined listing of this objects attributes (See the Get_Attributes_All Response definition below)
02 <sub>hex</sub>	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:

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Revision (low byte) Default = 1							

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
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 (see Chapter 4 of Volume I for a description of the Service Data field) of the Get\_Attributes\_All response is as follows:

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0								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 Default = 0
.								Owner Serial Number (high byte) Default = 0
.								Output Range Default = 255
.								Value Data Type Default = 0
.								Fault State Default = 0
.								Idle State 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:

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

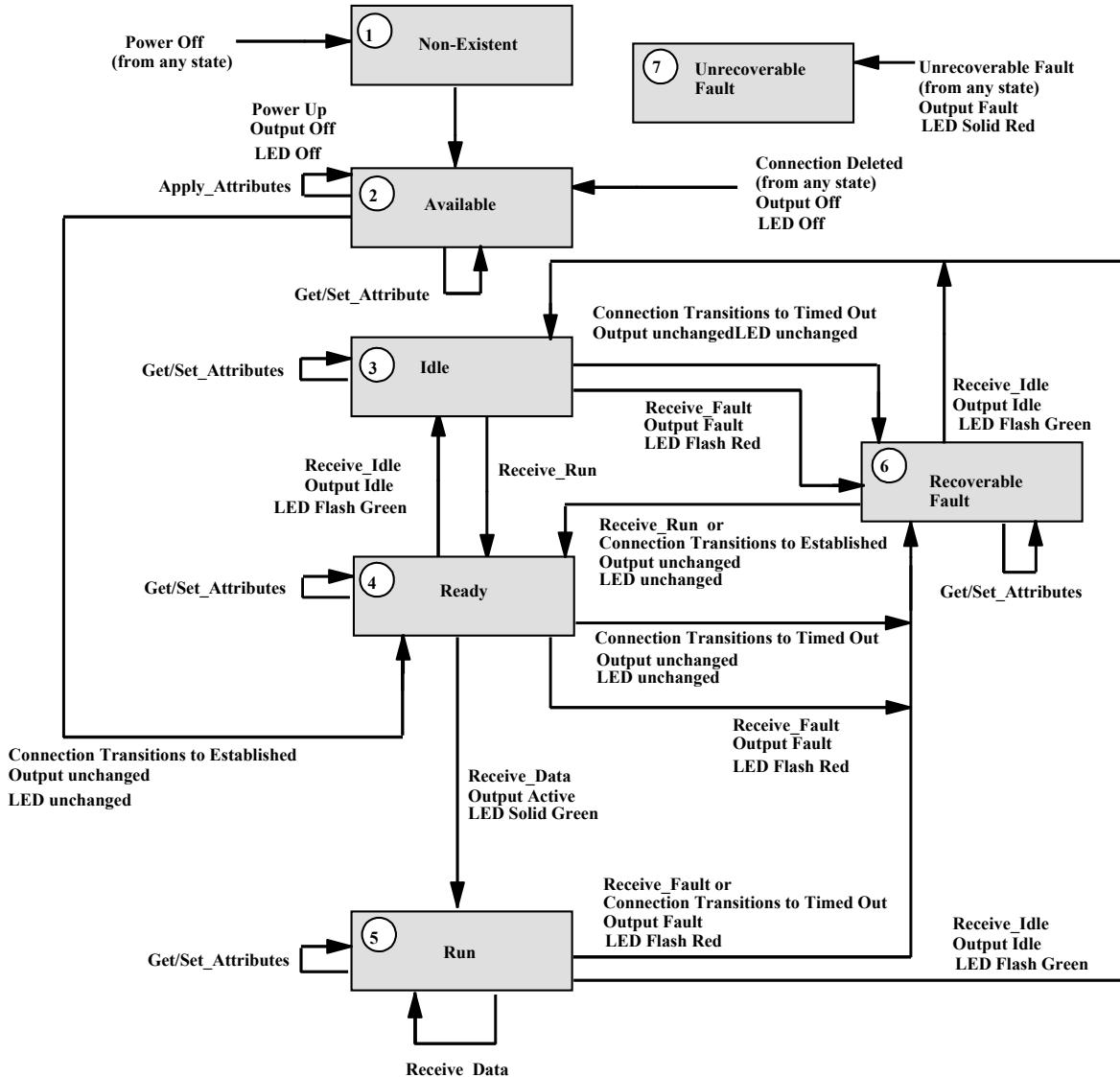
### 5-12.5. Behavior

The State Transition Diagram in Figure 5-12.1. 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 5-12.2 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.

Figure 5-12.1. State Transition Diagram for Analog Output Point Object



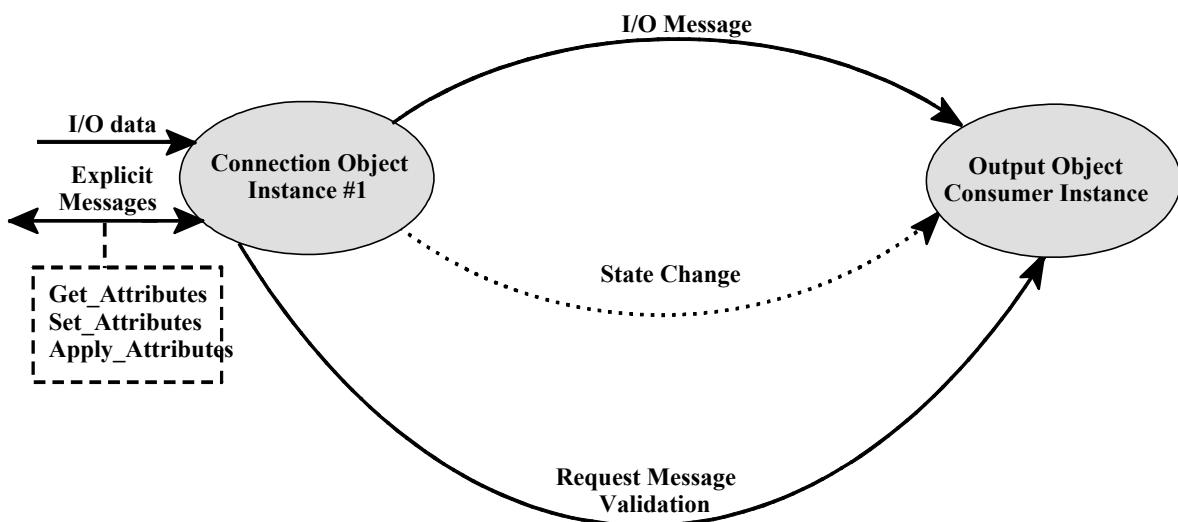
The following SEM contains these states:

- **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.

The SEM also contains these events:

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 <b>containing no application data</b>
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 <b>containing application data</b>
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.”



**Table 5-12.2. 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.

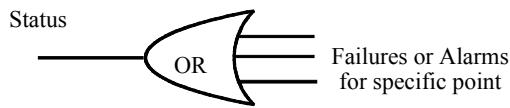
### Attribute Access Rules

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.

The optional *Status* attribute is simply a logical OR of all possible failure or alarm conditions for the point.



The *Command* attribute explicitly controls whether the AOP is in the Run or Idle state. It has effect only when the Discrete 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.

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

<b>This attribute pair</b>	<b>Defines</b>
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:

<b>If this attribute is supported</b>	<b>Then this attribute must also be supported by definition</b>
Output Fault State	Fault Value (although this attribute could be defined to always be LOW).
Output Idle State	Idle Value

The *Owner Vendor ID* and *Owner Serial Number* are included in the object definition but their use is TBD.

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).

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: 0Ehex

This object senses the presence or absence of a real world target.

### 5-13.1. 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

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

Semantics:

### 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.

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)

### Operate Mode

Use the Operate (Sensing) Mode to invert the level definition of the Output attribute.

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.

### 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).

### 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.

### 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.

### 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:

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	Optional	Set_Attributes_Single	Modifies an attribute value.

\*The Get\_Attribute\_Single service is REQUIRED if any attributes are implemented.

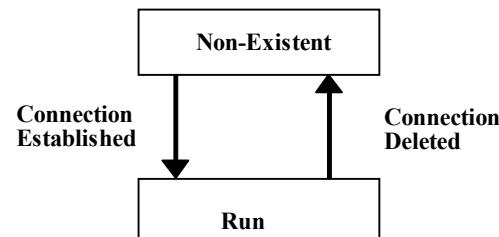
See the Appendix A for definitions of these common services.

### 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 the State Transition Diagram and State Event Matrix in this section. Table 5-13.1 indicates the accessibility of attributes.



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 5-13.1. Presence Sensing Object Attribute Access**

Attribute	State

	<b>Non-Existent</b>	<b>Running</b>
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: 0Fhex

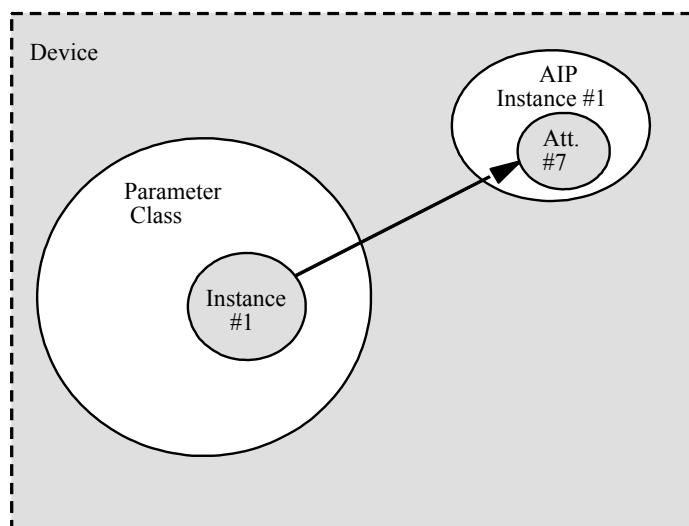
Use of the Parameter Object provides a known, public interface to a device's configuration data. In addition, this object also provides all the information necessary to define and describe each of a device's individual configuration parameters.

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.



### 5-14.1. 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 one (01).
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 6.J.
9	Required	Get	Configuration Assembly Instance	UINT	Instance number of the configuration assembly.	This attribute should be set to zero if a configuration assembly is not supported.
10	Optional	Set	Native Language	USINT	Language ID for all character array accesses.	0=English 1=French 2=Spanish 3=Italian 4=German 5=Japanese 6=Portugese 7=Mandarin Chinese

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

The Parameter Class Descriptor attribute contains bits to describe parameter characteristics. The following bits are supported:

**Table 5-14.1. 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

“*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:

Value	Meaning
1	Individual Parameter instances ARE supported.

	0      NO Parameter Instances are supported. Only configuration assembly instance used.
--	---

“*Supports Full Attributes*” indicates that each Parameter Instance contains all of the attributes and are not just stubs. Bit values are:

Value	Meaning
1	All Full Parameter Attributes ARE supported.
0	Only Parameter Instance stub attributes are supported.

“*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.

Value	Meaning
1	Must execute non-volatile storage save command.
0	Do not have to execute non-volatile storage save command.

“*Params are stored in Non-Volatile Storage*” indicates that parameters are stored in non-volatile storage. Bit values are:

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

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 6.K.
5	Required	Get	Stub	Data Type	EPATH	Data type code.	See Volume C, Section C-6.1

Attr ID	Need in Implementation	Access Rule	Stub/Full	Name	Data Type	Description of Attribute	Semantics of Values
6	Required	Get	Stub	Data Size	USINT	Number of bytes in Parameter Value	
7	Optional	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	Optional	Get	Full	Units String	SHORT_STRING	Engineering Unit String	The maximum number of characters is 4
9	Optional	Get	Full	Help String	SHORT_STRING	Help String	The maximum number of characters is 64
10	Optional	Get	Full	Minimum Value	<i>data type</i>	The minimum valid actual value to which the parameter can be set.	
11	Optional	Get	Full	Maximum Value	<i>data type</i>	The maximum valid actual value to which the parameter can be set.	
12	Optional	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	Optional	Get	Full	Scaling Multiplier	UINT	Multiplier for Scaling Factor.	See Figure 6.7.
14	Optional	Get	Full	Scaling Divisor	UINT	Divisor for Scaling Formula.	See Figure 6.7.
15	Optional	Get	Full	Scaling Base	UINT	Base for Scaling Formula.	See Figure 6.7.
16	Optional	Get	Full	Scaling Offset	INT (Can be negative)	Offset for Scaling Formula.	See Figure 6.7.
17	Optional	Get	Full	Multiplier Link	UINT	Parameter Instance of Multiplier source.	See Table 6.N.
18	Optional	Get	Full	Divisor Link	UINT	Parameter Instance of Divisor source.	See Table 6.N.
19	Optional	Get	Full	Base Link	UINT	Parameter Instance of Base source.	See Table 6.N.
20	Optional	Get	Full	Offset Link	UINT	Parameter Instance of Offset source.	See Table 6.N.
21	Optional	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.	

The Descriptor Instance Attribute contains these bits, which describe the parameter:

**Table 5-14.2. Semantics of Descriptor Instance Attribute**

Bit	Defintion	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.
4	Read Only Parameter	Indicates that the value attribute can only be read, and not set.
5	Monitor Parameter	Indicates that the value 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	Indicates that non-consecutive enumerated strings are supported.
8	Allows both enumeration and individual values	Both enumeration and individual values are supported.

The Data Type Instance Attribute specifies the data type of the parameter. This value shall be specified according to the format shown in Volume I, Appendix J, Section J-6.1. The following data types are obsolete but may be encountered in older devices:

**Table 5-14.3. 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

Attr Value	Definition	Description
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.

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 5-14.4. 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 5-14.5. 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:

$$\text{EngValue} = \frac{(\text{ActualValue} + \text{Offset}) * \text{Mult} * \text{Base}}{\text{Div} * 10^{\text{Precision}}}$$

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:

$$\text{ActualValue} = \frac{(\text{EngValue} * \text{Div} * 10^{\text{Precision}}) - \text{Offset}}{\text{Mult} * \text{Base}}$$

For the scaling formula, the following attributes are required:

**Table 5-14.6. Scaling Formula Attributes**

Attribute	Meaning
Multiplier Value	(Mult) Multiplier value for the scaling formula. This value can be based on another parameter.
Divisor Value	(Div) Divisor value for the scaling formula. This value can be based on another parameter specified in the Divisor Link.
Base Value	(Base) Base value for the scaling formula. This value can be based on another parameter specified in the Base Link.

Offset Value	(Offset) 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 5-14.7. Scaling Links**

Attribute	Meaning
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:

Service Code	Need in Implementation		Service Name	Description of Service
	Class	Instance		
0E <sub>hex</sub>	Required	Required	Get_Attribute_Single	Returns the contents of the specified attribute.
10 <sub>hex</sub>	Optional	Required	Set_Attribute_Single	Modifies an attribute value.
01 <sub>hex</sub>	Optional	Conditional	Get_Attributes_All	Returns a predefined listing of this objects attributes (See the Get_Attributes_All Response definition below)  Optional for Parameter Object Stubs, required for Full Parameter Objects
05 <sub>hex</sub>	Optional	n/a	Reset	Resets all parameter values to the factory default.
15 <sub>hex</sub>	Optional	n/a	Restore	Restores all parameter values from non-volatile storage.
16 <sub>hex</sub>	Optional	n/a	Save	Saves all parameter values to non-volatile storage.

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:

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:

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 1–21)

For Parameter object Stubs, 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:

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 (see Chapter 4 of Volume I for a description of the Service Data field) of the Get\_Attributes\_All response is as follows:

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
.								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

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 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

### 5-14.4. Object-specific Services

The Parameter Object provides the following Object-specific service:

Service Code	Need in Implementation		Service Name	Description of Service
4B <sub>hex</sub>	n/a	Optional	Get_Elem_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.

If the parameter's data type is:	Then the enumerated strings returned are:
WORD	<i>Bit</i> enumerated strings.
UINT or INT	<i>Value</i> enumerated strings.

For example, if the parameter's data type is WORD, requesting a Get\_Elem\_String service with a parameter of 0 returns a SHORT\_STRING that describes *bit* 0. Requesting a Get\_Elem\_String service with a parameter of 1 returns a SHORT\_STRING that describes *bit* 1.

If the parameter's data type is UINT or INT, requesting a Get\_Elem\_String service with a parameter of 0 returns a SHORT\_STRING that describes the *value* of 0. Requesting a Get\_Elem\_String service with a parameter of 1 returns a SHORT\_STRING that describes the *value* of 1.

The following parameters are defined for the Get\_Elem\_String service:

**Table 5-14.8. Parameters for Get\_Elem\_String Request**

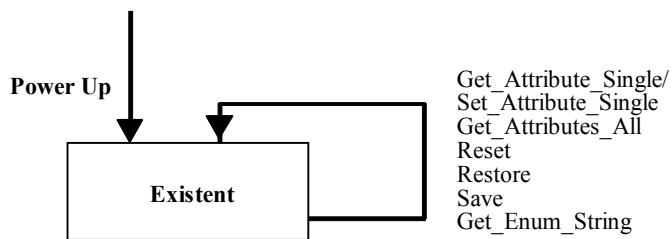
Name	Data Type	Description of Attribute	Semantics of Values
Enumerated String Number	USINT	Number of Enumerated String to retrieve.	

**Table 5-14.9. 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.



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: 10hex

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

Attribute 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	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

### 5-15.2. Instance Attributes

Attribute ID	Need in Implementation	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	

Attribute ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
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. Common Services

The Parameter Group Object provides the following Common Service:

Service Code	Need in Implementation		Service Name	Service Description
	Class	Instance		
0E <sub>hex</sub>	Required	Required	Get_Attribute_Single	Returns the contents of the specified attribute.
01 <sub>hex</sub>	Optional	Optional	Get_Attributes_All	Returns a predefined listing of this objects attributes (See the Get_Attributes_All Response definition below)
10 <sub>hex</sub>	Optional	n/a	Set_Attribute_Single	Modifies an attribute value.

#### 5-15.3.1. 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:

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							

At the Instance 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:

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)							

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
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)							

#### 5-15.4. Object-specific Services

The Parameter Group Object provides no Object-specific services on either the Class level or Instance level.

#### 5-15.5. Behavior

The behavior of the Parameter Group Object is illustrated in the State Transition Diagram and State Event Matrix below.

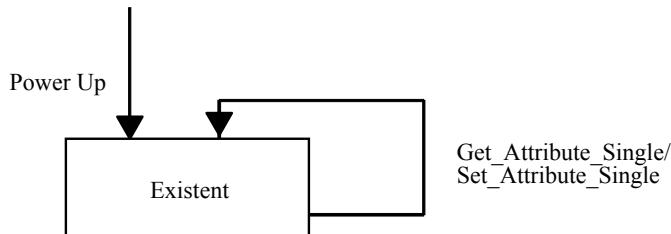


Table 5-15.1. State Event Matrix for Parameter Group Object

Event	State	
	Existent	
Get_Attribute_Single	Validates/services the request and returns the appropriate response.	
Set_Attribute_Single		

This page is intentionally left blank

## 5-16. GROUP OBJECT

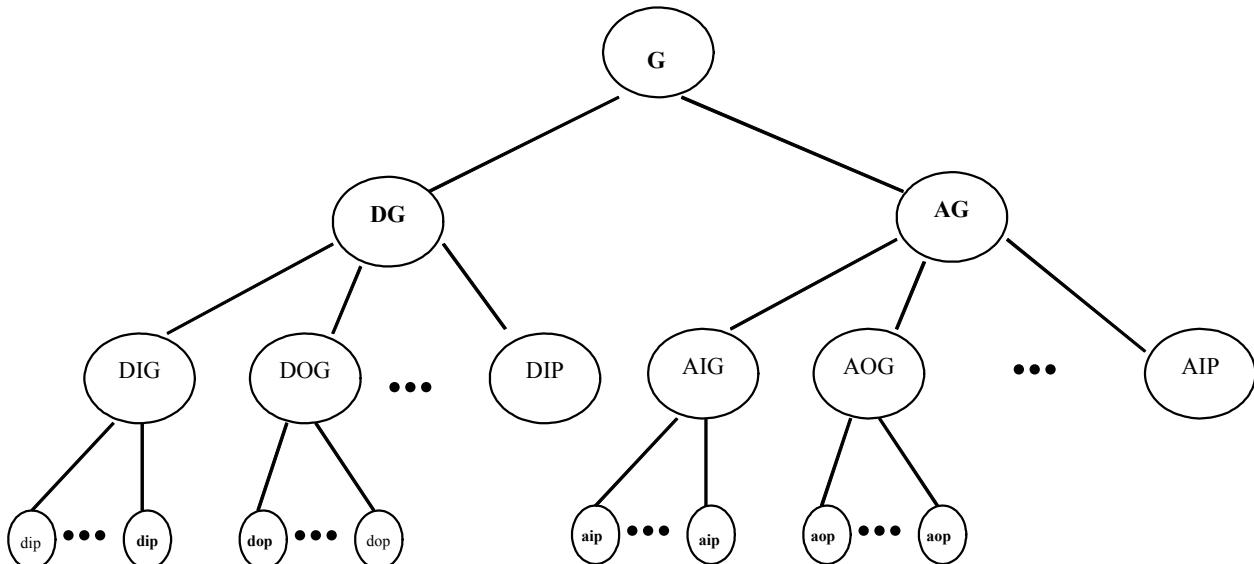
Class Code: 12hex

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.

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.



### 5-16.1. 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

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 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:

Service Code	Need in Implementation		Service Name	Description of Service
	Class	Instance		
0E <sub>hex</sub>	Optional*	Required	Get_Attribute_Single	Returns the contents of the specified attribute.
01 <sub>hex</sub>	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.

### 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 of Volume I for a description of the Service Data field) of the Get\_Attributes\_All response is as follows:

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							

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 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:

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

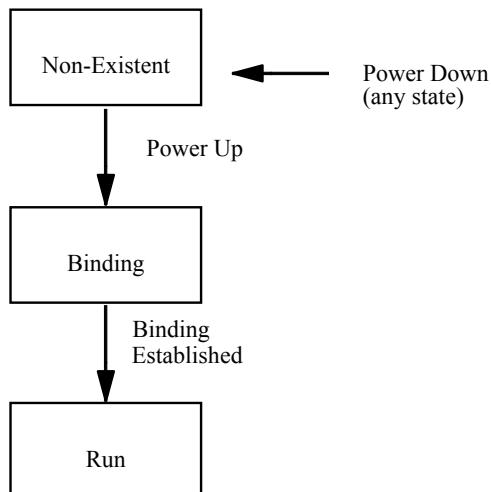
The Group Object provides no Object-specific services.

## 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:

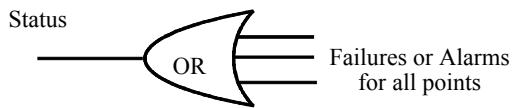
This state	Is equivalent to
Non-Existent	a module without power
	a module that has an unrecoverable fault (e.g. processor watchdog timeout)
	a major reconfiguration of the module (e.g. NVS Update)
Binding	when groups and/or points are added or bound to the Group (executed as part of Power Up cycle)
Run	the point at which Get and Set attribute services can be used to access the attributes of the Group Object



### Attribute Access Rules

All attributes are gettable or settable according to their attribute access rules, but only in the Run state.

The **Status** attribute is simply a logical OR of all possible failure or alarm conditions for all points bound to the group.



The Owner Vendor ID and Owner Serial Number are included in the object definition but their use is TBD.

## 5-17. DISCRETE INPUT GROUP OBJECT

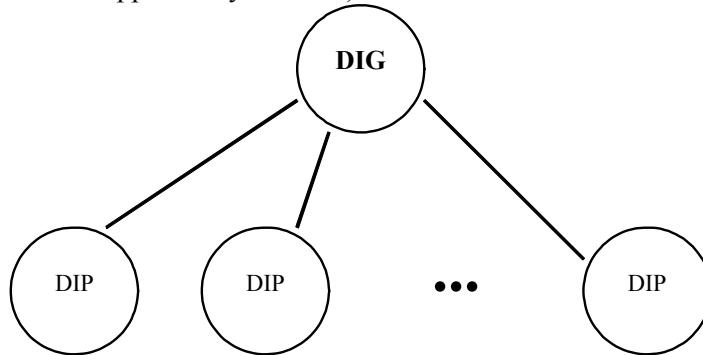
Class Code: 1Dhex

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.

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).



### 5-17.1. 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

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	

Attr ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
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. Common Services

The Discrete Input Group Object provides the following Common90 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>	Optional	Optional	Get_Attributes_All	Returns a predefined listing of this objects attributes (See the Get_Attributes_All Response definition below)
10 <sub>hex</sub>	n/a	Optional	Set_Attribute_Single	Modifies an attribute value.
02 <sub>hex</sub>	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.

#### 5-17.3.1. 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:

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:

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
.								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.

### 5-17.3.2. 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:

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.

### 5-17.4. Object-specific Services

The Discrete Input Group Object does not support any Object-specific services.

### 5-17.5. Behavior

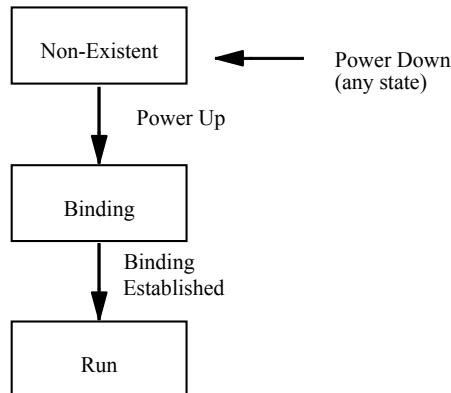
The primary purpose of the DIG is to bind discrete input points.

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.

The states shown are equivalent to the following:

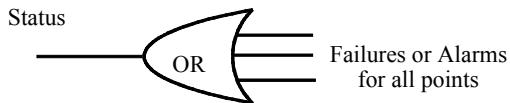
This state	Is equivalent to
Non-Existent	a module without power
	a module that has an unrecoverable fault (e.g. processor watchdog timeout)
	a major reconfiguration of the module (e.g. NVS Update)
Binding	when discrete input points are added or bound to the Discrete Input Group (executed as part of Power Up cycle)
Run	the point at which Get and Set attribute services can be used to access the attributes of the Group Object



### Attribute Access Rules

All attributes are gettable or settable according to their attribute access rules, but only in the Run state.

The **Status** attribute is simply a logical OR of all possible failure or alarm conditions for all points bound to the group.



## 5-18. DISCRETE OUTPUT GROUP OBJECT

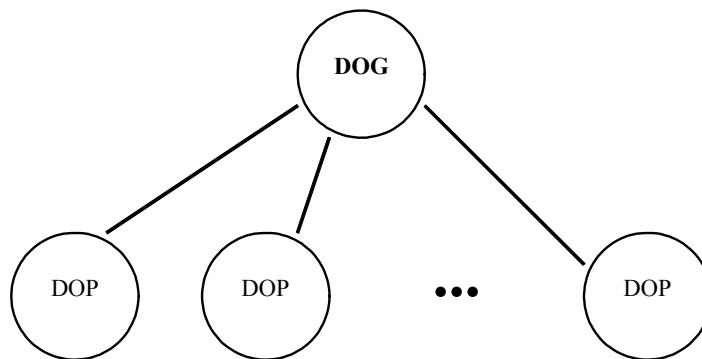
Class Code: 1Ehex

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.

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).



### 5-18.1. 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

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	

Attr ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
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 State	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 State	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. Common Services

The Discrete Output Group Object provides the following 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>	Optional	Optional	Get_Attributes_All	Returns a predefined listing of this objects attributes (See the Get_Attributes_All Response definition below)
10 <sub>hex</sub>	n/a	Required	Set_Attribute_Single	Modifies an attribute value.
02 <sub>hex</sub>	n/a	Optional	Set_Attribute_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 services.

#### 5-18.3.1. 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:

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 (see Chapter 4 of Volume I for a description of the Service Data field) of the Get\_Attributes\_All response is as follows:

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.

### 5-18.3.2. 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

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.

#### 5-18.4. Object-specific Services

The Discrete Output Group Object provides no Object-specific services.

#### 5-18.5. Behavior

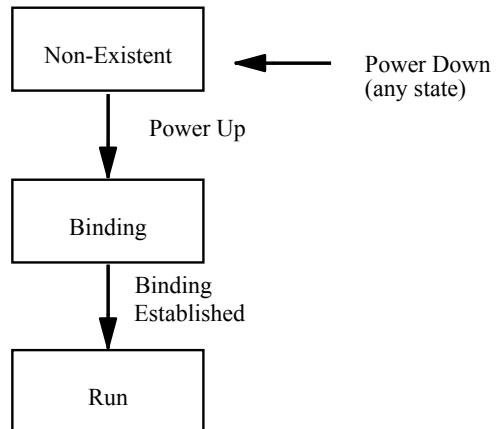
The primary purpose of the Discrete Output Group Object is to bind discrete output points.

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:

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



### Attribute Access Rules

All attributes are gettable or settable according to their attribute access rules, but only in the Run state.

The **Status** attribute is simply a logical OR of all possible failure or alarm conditions for all points bound to the group.

## 5-19. DISCRETE GROUP OBJECT

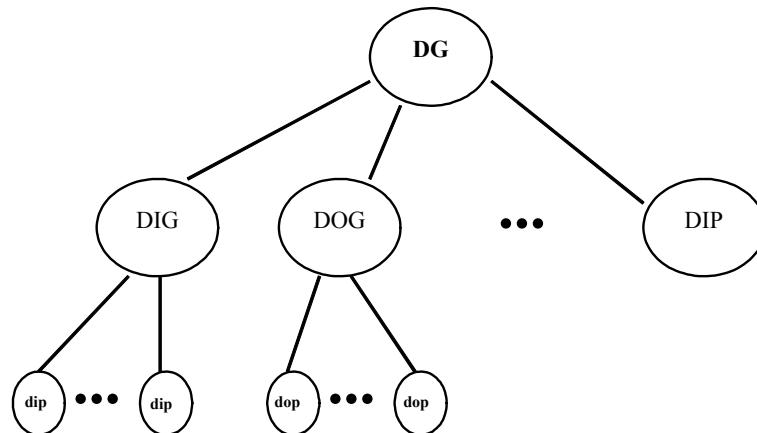
Class Code: 1Fhex

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.

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).



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

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

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 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:

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>	Optional	Optional	Get_Attributes_All	Returns a predefined listing of this objects attributes (See the Get_Attributes_All Response definition below)

\*The Get\_Attribute\_Single service is REQUIRED if any attributes are implemented.

### 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:

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 (see Chapter 4 of Volume I for a description of the Service Data field) of the Get\_Attributes\_All response is as follows:

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.

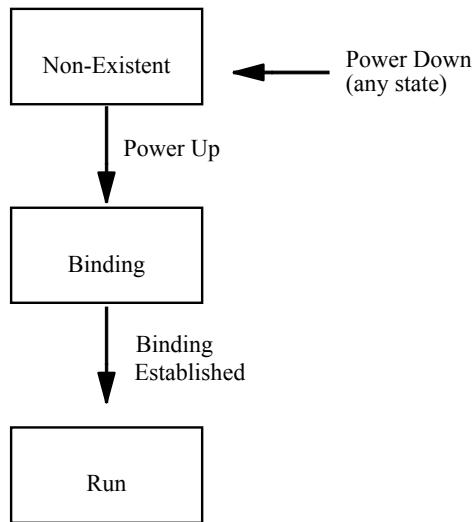
## 5-19.5. Behavior

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.

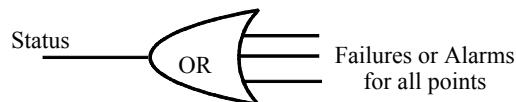
This state	Is equivalent to
Non-Existent	a module without power
	a module that has an unrecoverable fault (e.g. processor watchdog timeout)
	a major reconfiguration of the module (e.g. NVS Update)
Binding	when the discrete groups and/or points are added or bound to the Discrete Group (executed as part 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



### Attribute Access Rules

All attributes are gettable or settable according to their attribute access rules, but only in the Run state.

The **Status** attribute is simply a logical OR of all possible failure or alarm conditions for all points bound to the group.



## 5-20. ANALOG INPUT GROUP OBJECT

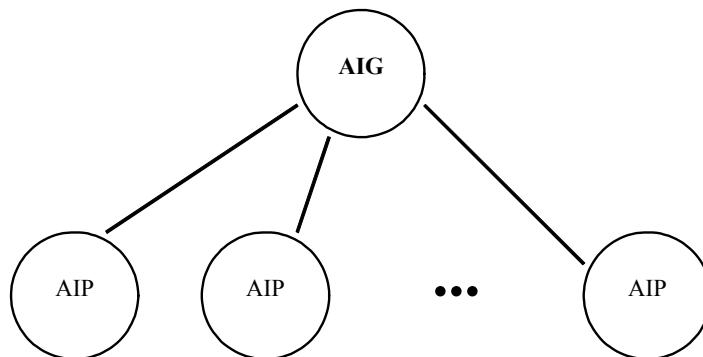
Class Code: 20hex

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.

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).



### 5-20.1. 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

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 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	

Attr ID	Need in implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
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. Common Services

The Analog Input Group Object provides the following Common Services:

Service Code	Need in Implementation		Service Name	Description of Service
	Class	Instance		
0E <sub>hex</sub>	Conditional <sup>1</sup>	Required	Get_Attribute_Single	Returns the contents of the specified attribute.
10 <sub>hex</sub>	n/a	Required <sup>2</sup>	Set_Attribute_Single	Modifies an attribute value.
01 <sub>hex</sub>	Optional	Optional	Get_Attributes_All	Returns a predefined listing of this objects attributes (See the Get_Attributes_All Response definition below)
02 <sub>hex</sub>	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 the Appendix A for definitions of these services.

### 5-20.3.1. 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:

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 (see Chapter 4 of Volume I for a description of the Service Data field) of the Get\_Attributes\_All response is as follows:

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.

### 5-20.3.2. 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 (see Chapter 4 of Volume I for a description of the Service Data field) of the Set\_Attributes\_All request is as follows:

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.

### 5-20.4. Object-specific Services

The Analog Input Group Object provides no Object-specific services.

### 5-20.5. Behavior

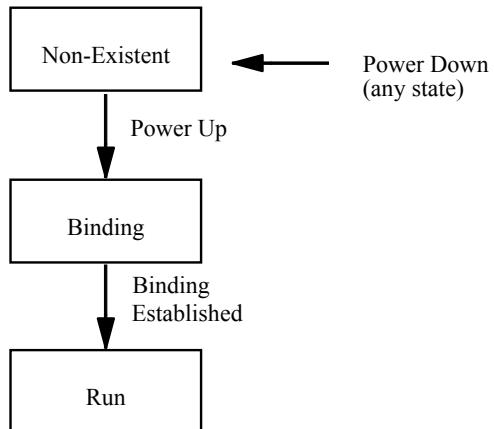
The primary purpose of the Analog Input Group Object is to bind analog input points.

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:

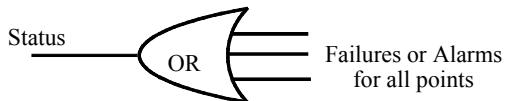
This state	Is equivalent to
Non-Existent	a module without power
	a module that has an unrecoverable fault (e.g. processor watchdog timeout)
	a major reconfiguration of the module (e.g. NVS Update)
Binding	when analog input points are added or bound to the Analog Input Group (executed as part of Power Up cycle)
Run	the point at which Get and Set attribute services can be used to access the attributes of the Group Object



### Attribute Access Rules

All attributes are gettable or settable according to their attribute access rules.

The **Status** attribute is simply a logical OR of all possible failure or alarm conditions for all points bound to the group.



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.

The **Owner Vendor ID** and **Owner Serial Number** are included in the object definition but their use is TBD.

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

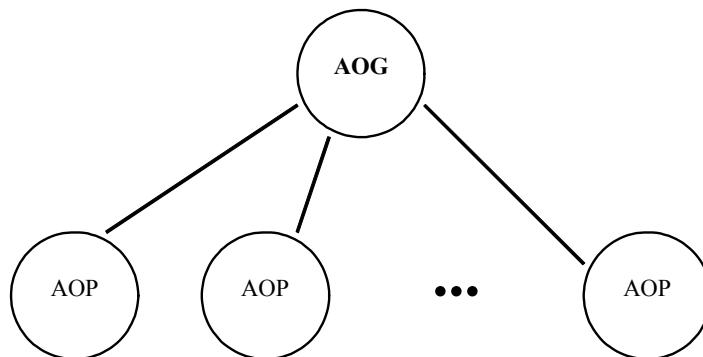
Class Code: 21hex

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.

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).



### 5-21.1. 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.					

### 5-21.2. Instance Attributes

Attr ID	Need in Implementation	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	

Attr ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
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 any bound Values	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 State	BOOL	State of output after recoverable failure	0=Fault Value attribute; 1=hold last state
11	Optional	Set	Fault Value	INT or based on attribute 8	User-defined value for use with Fault State attribute	
12	Optional	Set	Idle State	BOOL	State of output during idle	0=Idle Value attribute; 1=hold last state
13	Optional	Set	Idle Value	INT or based on attribute 8	User-defined value for use with Idle State attribute	

### 5-21.3. Common Services

The Analog Output Group Object provides the following Common Services:

Service Code	Need in Implementation		Service Name	Description of Service
	Class	Instance		
0E <sub>hex</sub>	Conditional <sup>1</sup>	Required	Get_Attribute_Single	Returns the contents of the specified attribute.
10 <sub>hex</sub>	n/a	Required <sup>2</sup>	Set_Attribute_Single	Modifies an attribute value.
01 <sub>hex</sub>	Optional	Optional	Get_Attributes_All	Returns a predefined listing of this objects attributes (See the Get_Attributes_All Response definition below)
02 <sub>hex</sub>	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 Attribute #8 is implemented.

See Appendix A for definitions of these services.

### 5-21.3.1. 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:

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:

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 Default = 0
.								Owner Serial Number (high byte) Default = 0

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

### 5-21.3.2. 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:

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.

### 5-21.4. Object-specific Services

The Analog Output Group Object provides no Object-specific services.

## 5-21.5. Behavior

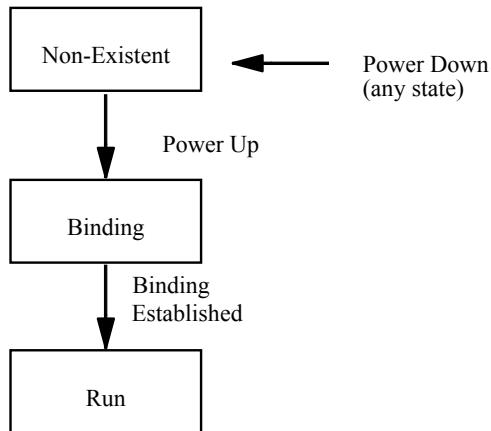
The primary purpose of the Analog Output Group is to bind Analog Output Points.

An attribute of the AOG will modify the behavior or report additional status information of all AOPs bound to the group.

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:

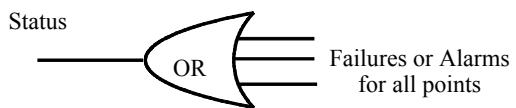
This state	Is equivalent to
Non-Existent	a module without power
	a module that has an unrecoverable fault (e.g. processor watchdog timeout)
	a major reconfiguration of the module (e.g. NVS Update)
Binding	when analog output points are added or bound to the Analog Output Group (executed as part of Power Up cycle)
Run	the point at which Get and Set attribute services can be used to access the attributes of the Group Object



### Attribute Access Rules

All attributes are gettable or settable according to their attribute access rules, but only in the Run state.

The **Status** attribute is simply a logical OR of all possible failure or alarm conditions for all points bound to the group.



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.

The **Owner Vendor ID** and **Owner Serial Number** are included in the object definition but their use is TBD.

## 5-22. ANALOG GROUP OBJECT

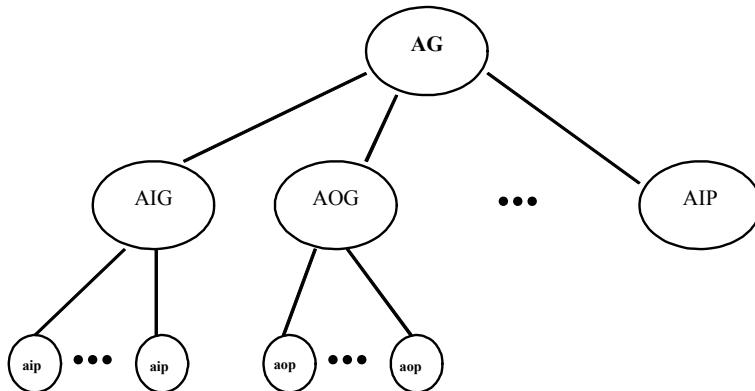
Class Code: 22hex

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.

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).



### 5-22.1. 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

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 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:

Service Code	Need in Implementation		Service Name	Description of Service
	Class	Instance		
0E <sub>hex</sub>	Conditional <sup>1</sup>	Required	Get_Attribute_Single	Returns the contents of the specified attribute.
10 <sub>hex</sub>	n/a	Required <sup>2</sup>	Set_Attribute_Single	Modifies an attribute value.
01 <sub>hex</sub>	Optional	Optional	Get_Attributes_All	Returns a predefined listing of this objects 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 Attribute #8 is implemented.

#### 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:

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

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 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 (see Chapter 4 of Volume I for a description of the Service Data field) of the Get\_Attributes\_All response is as follows:

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
.								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

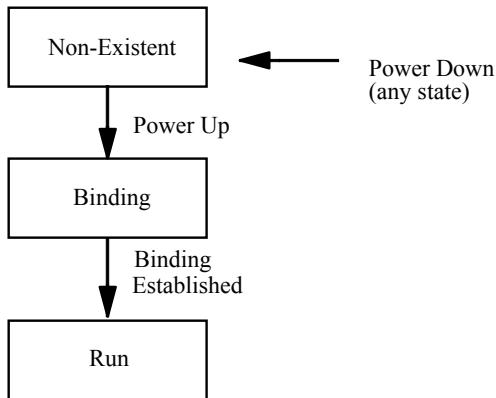
The primary purpose of the Analog Group Object is to bind analog groups and/or points.

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:

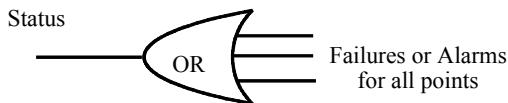
This state	Is equivalent to
Non-Existent	a module without power a module that has an unrecoverable fault (e.g. processor watchdog timeout) a major reconfiguration of the module (e.g. NVS Update)
Binding	when the analog input and/or output groups and/or points are added or bound to the Analog Group (executed 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



### Attribute Access Rules

All attributes are gettable or settable according to their attribute access rules, but only in the Run state.

The **Status** attribute is simply a logical OR of all possible failure or alarm conditions for all points bound to the group.



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.

The **Owner Vendor ID** and **Owner Serial Number** are included in the object definition but their use is TBD.

## 5-23. POSITION SENSOR OBJECT

Class Code: 23hex

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. Class Attributes

Attr 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.					

### 5-23.2. 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 in this product	
2	Optional	Get	Attribute List	Array of USINT	List of attributes supported in this product.	
3	Required	Get	Value	UDINT	Current position.	Physical position modified by the Resolution and Zero Offset Attributes.
4	Optional	Get	CAM	BOOL	Virtual CAM switch value	0 = Off 1 = On
5	Optional	Get/Set	Value Bit Resolution	USINT	Position sensor resolution. May be greater or less than the physical position sensor resolution.	The useable resolution of the position sensing device. The default is the physical resolution of the position sensing hardware. (valid range is 1 to 32) <i>i.e. Setting this value to 7 limits Value to 0 to 127 (Before Zero Offset is applied).</i>

Attr ID	Need in implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
6	Optional	Get/Set	Zero Offset	UDINT	Value attribute zero offset, subject to Resolution attribute.	This attribute offsets the Value attribute after the Resolution Attribute has been applied. This effectively sets the zero point for the Value attribute. <i>The default is 0.</i>
7	Optional	Get/Set	CAM Low Limit	UDINT	Virtual CAM switch low limit.	<i>The default is 0</i>
8	Optional	Get/Set	CAM High Limit	UDINT	Virtual CAM switch high limit.	<i>The default is 0</i>
9	Optional	Get/Set	SetZero	BOOL	Auto zero control.	<i>A rising edge on this signal sets Zero Offset to the current position.</i>

### 5-23.3. Common Services

The Position Sensor Object provides the following 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	Optional	Set_Attribute_Single	Modifies an attribute value.

\* The Get\_Attribute\_Single service is *required* at the class level if any class attributes are implemented

### 5-23.4. Object-specific Services

The Position Sensor Object provides no Object-specific services.

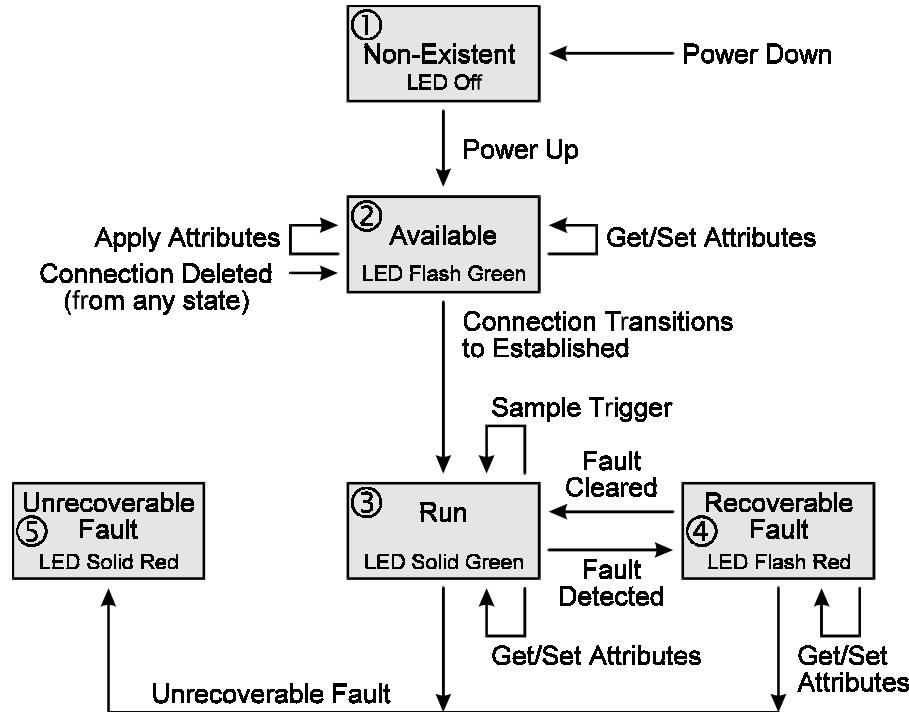
### 5-23.5. Behavior

The State Transition Diagram (Figure 5-23.1) 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 (See Table 5-23.2) 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.

### 5-23.1. State Transition Diagram for Position Sensor Object



LED = I/O Status LED

**Important:** Events can occur simultaneously, but the *Fault* events have priority if they occur simultaneously with other events.

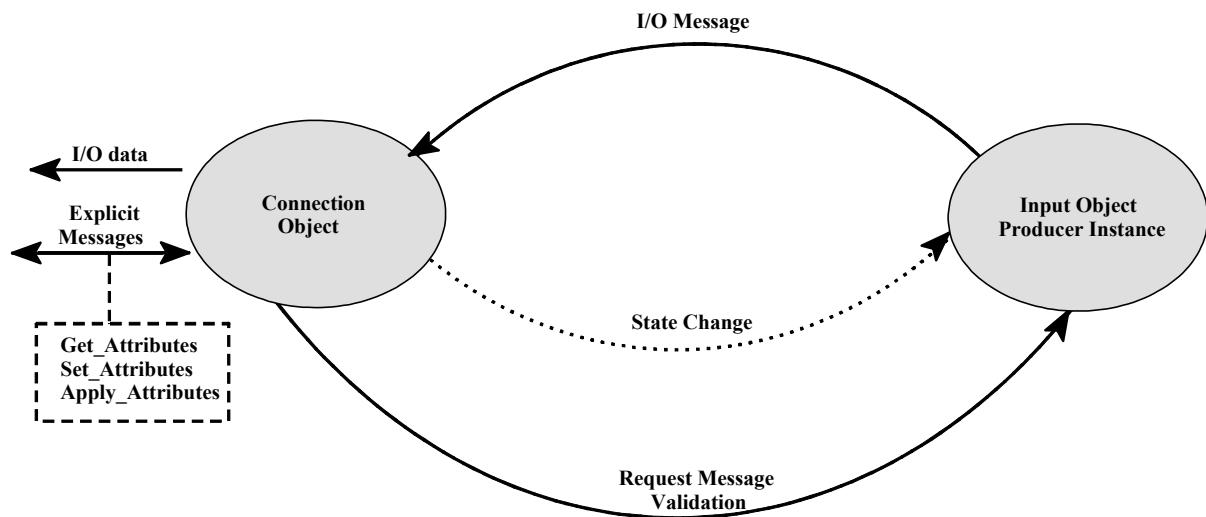
The following SEM contains these states:

- **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:

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.

This event	Is
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.”



**Table 5-23.2. 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 <b>Available</b>	Transition to <b>Available</b>	Ignore event
Connection Transitions to Established	Not Applicable	Transition to <b>Run</b>	Ignore event	Ignore event	Ignore event
Connection Transitions to Timed Out state	Not Applicable	Ignore event	Transition to <b>Recoverable Fault</b>	Ignore event	Ignore event
Fault Cleared	Not Applicable	Not Applicable	Not Applicable	Transition to <b>Run</b>	Ignore event
Get_Attribute	Return Error (Object Does Not Exist)	Return value	Return value	Return value	Ignore event

Event	State				
	Non-Existent	Available	Run	Recoverable Fault	Unrecoverable Fault
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

### Attribute Access Rules

Except in the Non-Exist and Unrecoverable Fault states, all attributes are gettable or settable according to their access rules.

#### Value

The Value Attribute represents the absolute position detected by the position sensor conditioned by the Resolution and Zero Offset attributes. Refer to the following descriptions of the Resolution and Zero offset attributes for details.

#### Bit Resolution

The Bit Resolution Attribute specifies the number of significant bits used for Value. The raw value is shifted left or right to supply the indicated number of significant bits.

Resolution	Value =
> Physical Resolution	$(RawValue << (PhysicalResolution - Bit Resolution)) + ZeroOffset$
< Physical Resolution	$(RawValue >> (Bit Resolution - PhysicalResolution)) + ZeroOffset$
= Physical Resolution	$RawValue + ZeroOffset$

#### Example

Raw Value	Resolution	Adjusted Val.	Notes
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.

#### Zero Offset

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 Resolution Attribute.

If the result of the addition exceeds the maximum specified by the Resolution attribute the overflow bits are discarded.

#### Example

Adjusted Val.	Zero Offset	Resolution	Value
0	10	8	10
250	0	8	250

Adjusted Val.	Zero Offset	Resolution	Value
250	20	8	15 <sup>1</sup>
250	20	10	270
250	255	8	249 <sup>2</sup>

<sup>1</sup> Value overflowed<sup>2</sup> Value underflowed

### Auto Zero

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 Value attribute being zero.

If the Zero Offset attribute is implemented as non-volatile, the AutoZero command must store the new Zero Offset value.

### CAM, CAM Low Limit, CAM High Limit

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 Value attributes. The Value Attribute is used after the Resolution and Zero Offset Attributes have been applied.

CAM Low Limit	CAM is On (true, 1) if ...	CAM is Off (false, 0) if ...
> 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

## 5-24. POSITION CONTROLLER SUPERVISOR OBJECT

Class Code: 24hex

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:

Revision	Description
01	Initial release
02	1. Class Attributes 32 and 33 added

### 5-24.2. 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 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 Section 6-12.5: I/O Connection Messages of the Position Controller Device Profile 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 Section 6-12.5: I/O Connection Messages of the Position Controller Device Profile 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 Position Controller Supervisor Object Instance Attributes:

### 5-24.3.1 Supervisor Attributes:

Attr ID	Need in Implementation	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.

Attr ID	Need in Implementation	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. Action codes 128 through 255 are for vendor specific action.

### 5-24.3.2 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. Action codes 128 through 255 are for vendor specific action.
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. Action codes 128 through 255 are for vendor specific action.
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 0xFFFFFFFF.
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 0xFFFFFFFF.

### 5-24.3.3 Registration Attributes

Attr ID	Need in Implementation	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. Action codes 128 through 255 are for vendor specific action.
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 0xFFFFFFFF.
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 0xFFFFFFFF.

### 5-24.3.4 Axis Following Attributes:

Attr ID	Need in Implementation	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.	

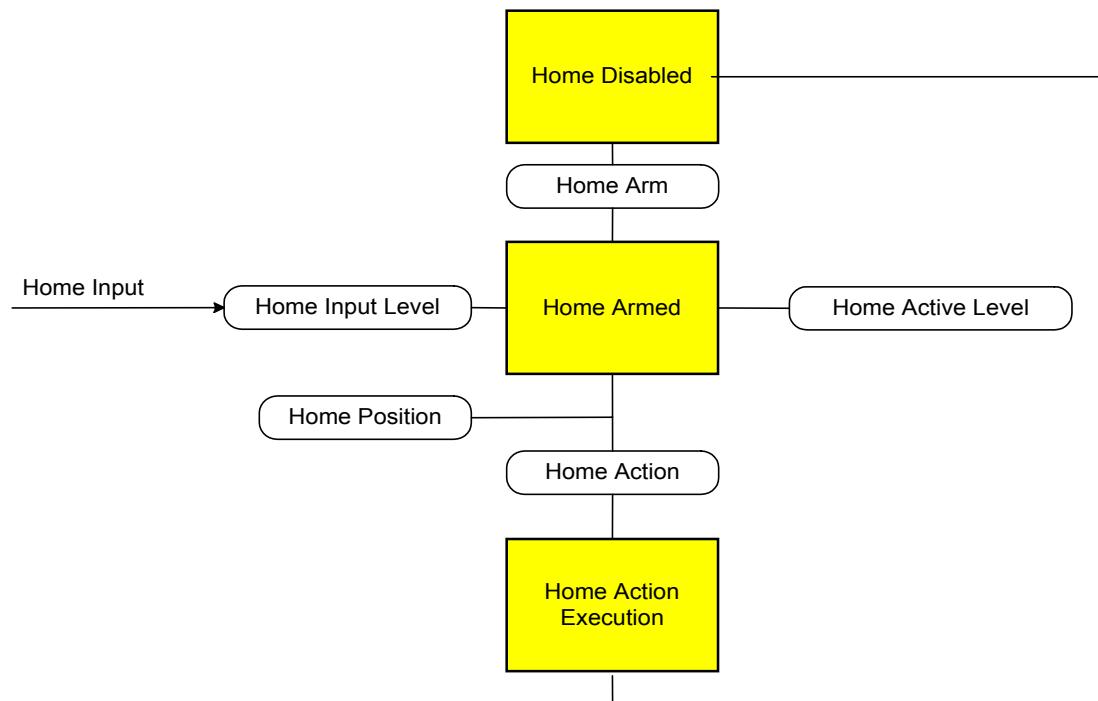
## 5-24.4. Position Controller Supervisor Supported Services:

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.

## 5-24.5. Position Controller Supervisor State Diagrams

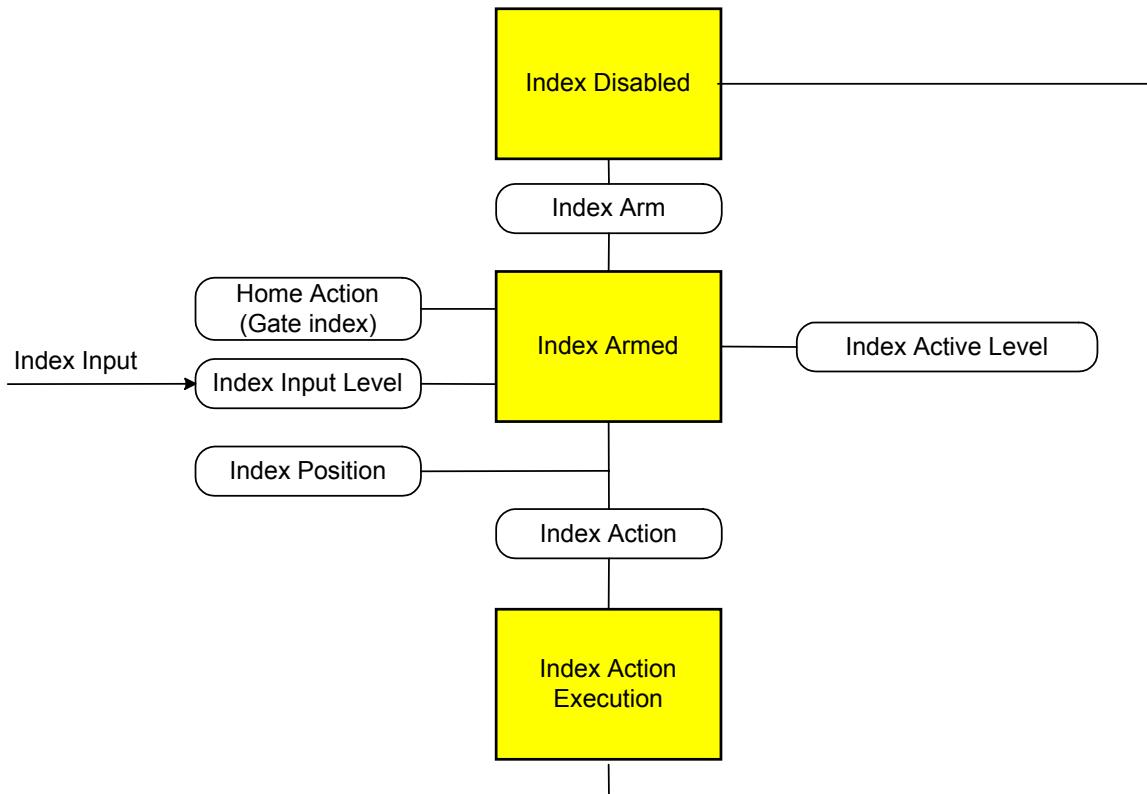
### 5-24.5.1. Home Input State Diagrams

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.



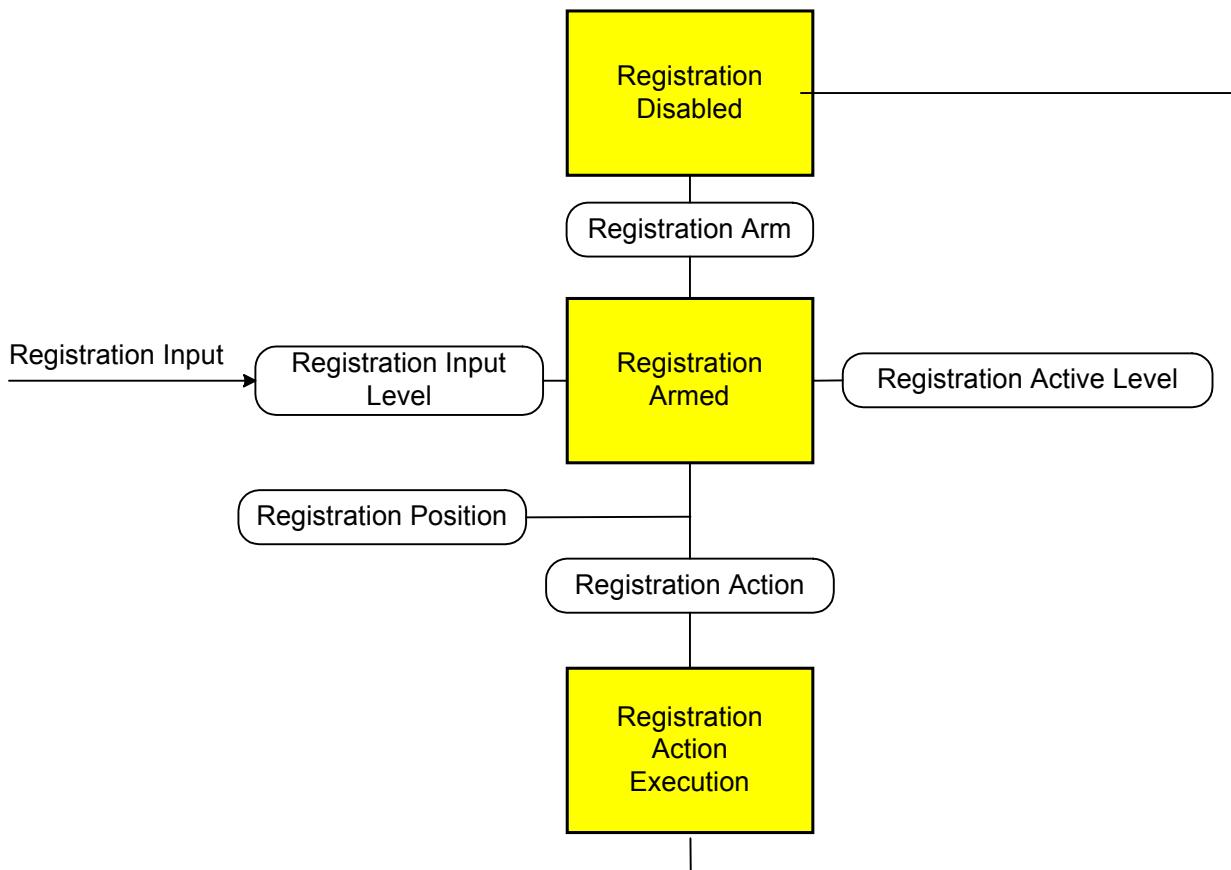
### 5-24.5.2 Index Input State Diagram

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.



### 5-24.5.3. 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.



## 5-25. POSITION CONTROLLER OBJECT

Class Code: 25hex

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:

Revision	Description
01	Initial release
02	1. Instance Attribute 58 added

### 5-25.2 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 either optional or conditional and are described in Chapter 4 of this specification.					

### 5-25.3 Position Controller Object Instance Attributes:

Attr ID	Need in Implementation	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.

Attr ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
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 0x7FFFFFFF.
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.
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.

Attr ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
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. 128 to 255 = vendor specific profile types.
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.
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.

Attr ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
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	Propotional 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.

Attr ID	Need in Implementation	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. Action codes 128 through 255 are for vendor specific action.
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.

Attr ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
49	Optional	Set	Hard Limit Action	USINT	Hard limit action code.	0 = Command Output Generator Off 1 = Hard Stop 2 = smooth stop. Action codes 128 through 255 are for vendor specific action.
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.
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.

### Semantics:

#### 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.

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 3-12.4.3 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 Sinble	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.

### 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 messagedata bytes. This attribute will be reset when the Load Command Data/Start Profile bit is reset. Refer to Section 3-12.4.4 for an explanation of the handshaking procedure. This bit is not affected by explicit messaging.

## 5-25.4. Position Controller Supported 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.

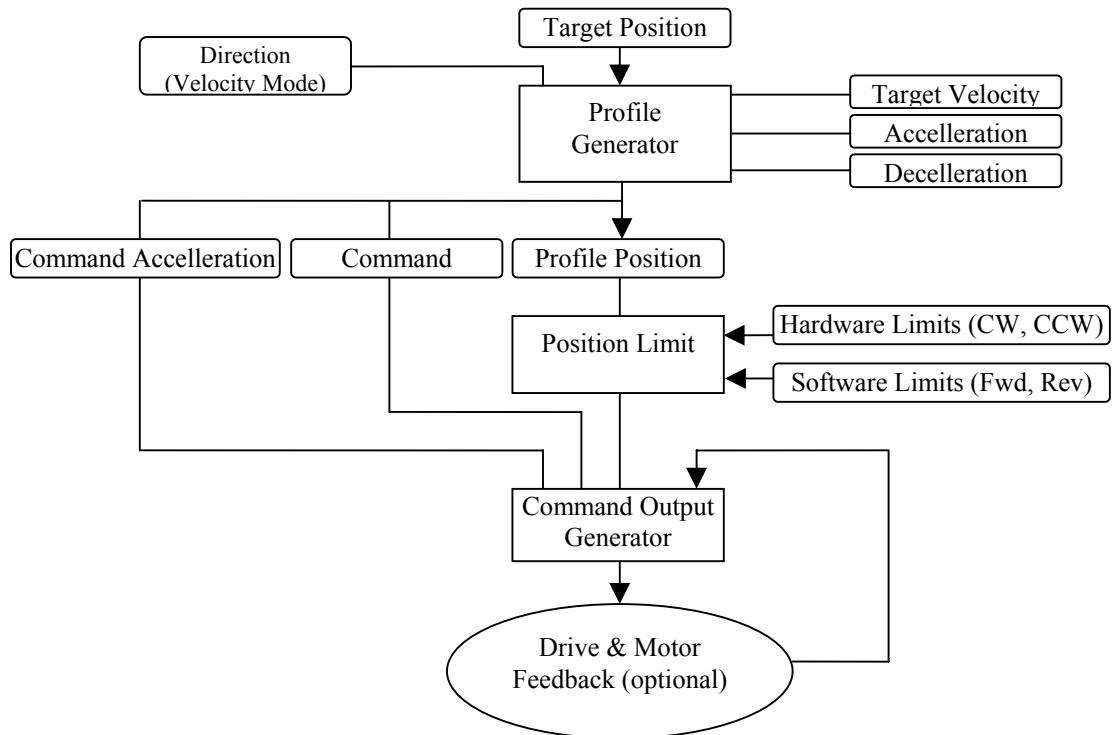
## 5-25.5. Position Controller State Diagrams

### 5-25.5.1. 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.

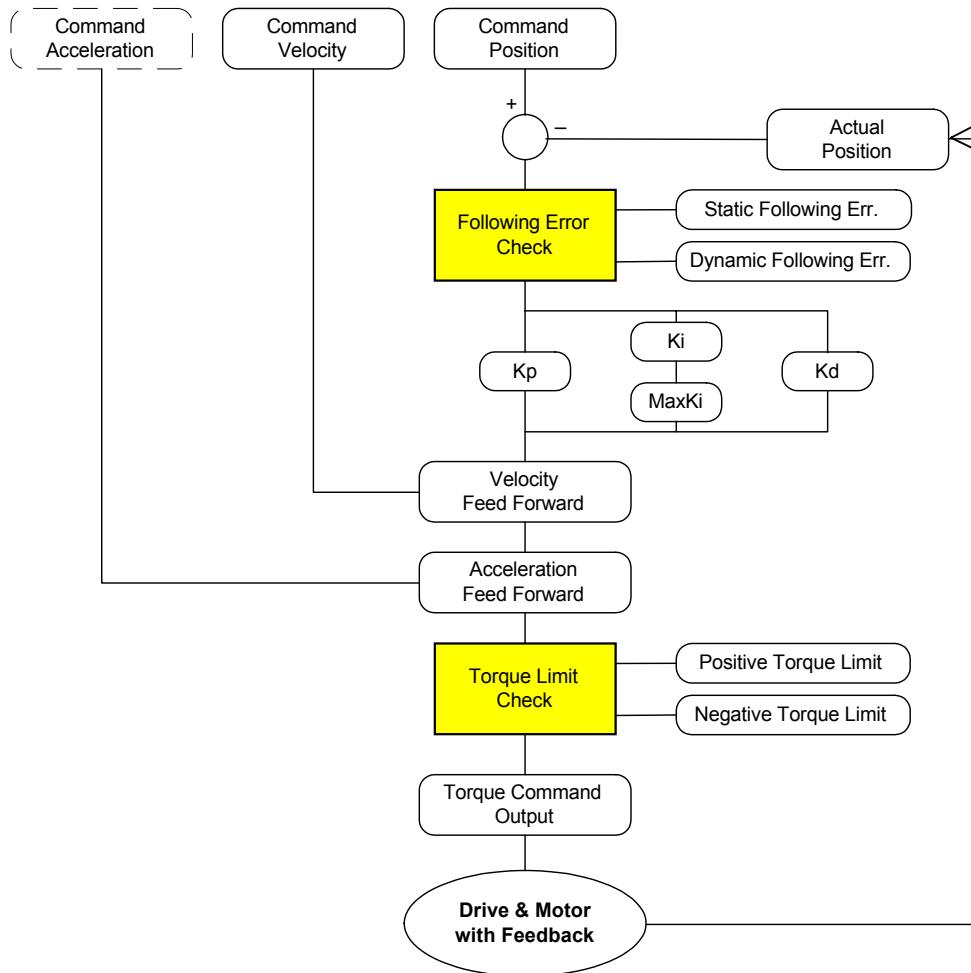
After the profile position is generated, it passes through a limit filter. If the generated Profile position is outside the defined software limits, or if a hardware limit is active, the profile is modified and the output Command position is limited. The limit function and attributes are optional.

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.



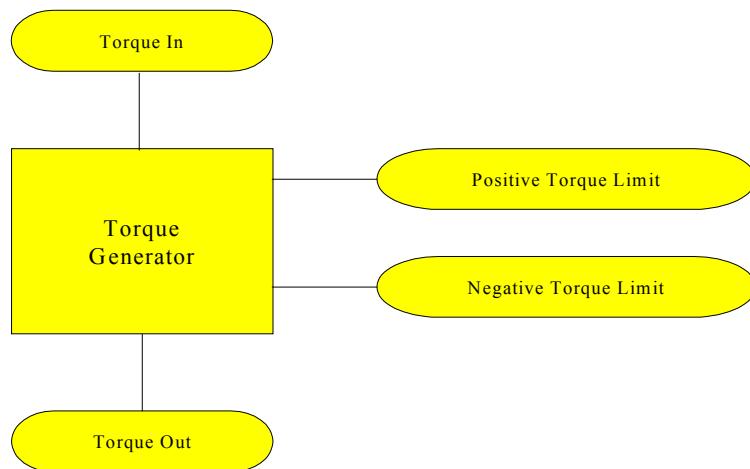
### 5-25.5.2. Servo Output Generation State Diagram

The diagram below is an example of a Servo Command output generator. Commanded Position, Velocity and Acceleration input comes from the Profile Generation diagram.



### 5-25.5.3. Torque Mode Output State Diagram

The following diagram describes the direct torque mode function.



## 5-26. BLOCK SEQUENCER OBJECT

Class Code: 26hex

This object handles the execution of Command Blocks or Command Block chains.

### 5-26.1. 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. Block Sequencer Object Instance Attributes:

These attributes can be used for control, configuration or status.

Attr ID	Need in Implementation	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.

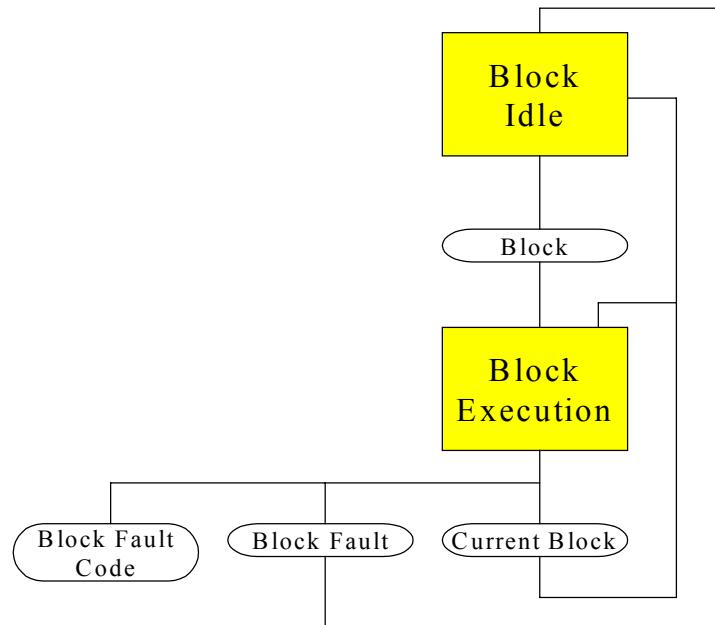
Attr ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
5	Optional	Get	Block Fault Code	BOOL	Block fault Code.	Defines the specific block fault. 0 = no fault, 1 = invalid or empty block data, 2 = command time-out (Wait Equals), 3 = execution fault.
6	Optional	Set	Counter	DINT	Sequencing Counter.	Must be positive. Counter that can be used for sequencing loops.

### 5-26.3. Block Sequencer Supported 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.

### 5-26.4. 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.



## 5-27. COMMAND BLOCK OBJECT

Class Code: 27hex

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

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. Command Block Object Instance Attributes:

These attributes are down-loaded 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.

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 attributes value. Attribute 1 must be set to 01.

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.

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.

Attribute 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 Attribute	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.

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.

Attr ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
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.

Attribute 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.

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.

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.

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.

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.

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.

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.

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. Command Block Supported 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.

## 5-28. MOTOR DATA OBJECT

Class Code: 28hex

This object serves as a database for motor parameters.

### 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. Motor Data 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.

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	Set/Get	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	CatNumber	SHORT_STRING	Manufacturer's Motor Catalog Number (Nameplate number) 32 chars max
5	Optional	Set/Get	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.

<b>Motor Class</b>	<b>Motor Types in Class</b>	<b>Section Reference</b>
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

<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>
6	Required	Set/Get	RatedCurrent	UINT	Rated Stator Current Units: [100mA]
7	Required	Set/Get	RatedVoltage	UINT	Rated Base Voltage Units: [V]
8	Optional	Set/Get	RatedPower	UDINT	Rated Power at Rated Freq Units: [W]
9	Optional	Set/Get	RatedFreq	UINT	Rated Electrical Frequency Units: [Hz]
10	Optional	Set/Get	RatedTemp	UINT	Rated Winding Temperature Units: [ degrees C]
11	Optional	Set/Get	MaxSpeed	UINT	Maximum allowed motor speed Units: [RPM]
12	Optional	Set/Get	PoleCount	UINT	Number of poles in the motor.
13	Optional	Set/Get	TorqConstant	UDINT	Motor torque constant Units: [0.001 x Nm/A]
14	Optional	Set/Get	Inertia	UDINT	Rotor Inertia Units: [ $10^{-6}$ x kg.m $^2$ ]
15	Optional	Set/Get	BaseSpeed	UINT	Nominal speed at rated frequency from nameplate Units: [RPM]
19	Optional	Set/Get	ServiceFactor	USINT	Units: [%] Range: 0 .. 255

### 5-28.2.1.2. DC Motor Instance Attributes

Attribute ID	Need in implementation	Access Rule	Name	Data Type	Description of Attribute
6	Required	Set/Get	RatedCurrent	UINT	Rated Armature Current Units: [100mA]
7	Required	Set/Get	RatedVoltage	UINT	Rated Armature Voltage Units: [V]
8	Optional	Set/Get	RatedPower	UDINT	Rated Power at MaxSpeed Units: [W]
10	Optional	Set/Get	RatedTemp	UINT	Rated Winding Temperature Units: [ degrees C]
11	Optional	Set/Get	MaxSpeed	UINT	Maximum allowed motor speed Units: [RPM]
13	Optional	Set/Get	TorqConstant	UDINT	Motor torque constant Units: [0.001 x Nm/A]
14	Optional	Set/Get	Inertia	UDINT	Rotor Inertia Units: [ $10^{-6} \times \text{kg.m}^2$ ]
15	Optional	Set/Get	BaseSpeed	UINT	Nominal speed at rated voltage Units: [RPM]
16	Optional	Set/Get	RatedFieldCur	UDINT	Rated Field Current Units: [mA]
17	Optional	Set/Get	MinFieldCur	UDINT	Minimum Field Current Units: [mA]
18	Optional	Set/Get	RatedFieldVolt	UINT	Rated Field Voltage Units: [V]

The following engineering abbreviations are used in the above Motor Data Instance Attribute descriptions.

Abbreviation	Description
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. Motor Data 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.

Service Code	Need in Implementation		Service Name	Description of Service
	Class	Instance		
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. Motor Data Object-specific Services

The Motor Data object provides no object specific services.

#### 5-28.5. Motor Data Object 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: 29hex

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. Control Supervisor 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. Control Supervisor 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	Set/Get	Run1	BOOL	See Run/Stop Event Matrix
4	Optional	Set/Get	Run2	BOOL	See Run/Stop Event Matrix
5	Optional	Set/Get	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
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

<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>
12	Required for Drives and Servos only	Set/Get	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/Get	DNFaultMode	USINT	Action on loss of CIP Network 0 = Fault + Stop 1 = Ignore (Warning Optional) 2= Vendor specific
17	Optional	Set/Get	ForceFault/Trip	BOOL	0 ->1 = Force
18	Optional	Get	ForceStatus	BOOL	0 = Not Forced Nonzero = Forced

### **5-29.3. Control Supervisor 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 drive to the <b>start-up</b> state.

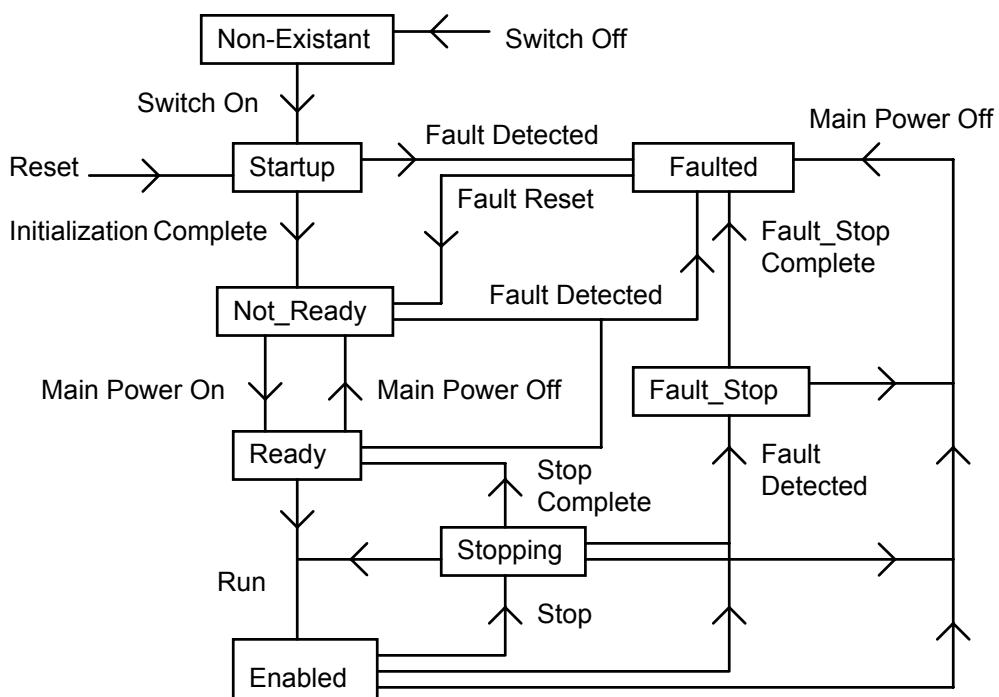
#### **5-29.4. Control Supervisor Object Specific Services**

The Control Supervisor object provides no object specific services.

### **5-29.5. Control Supervisor 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.

## Control Supervisor State Transition Diagram



### Control Supervisor State Event Matrix

Event	State							
	Non_Exist	Startup	Not_Ready	Ready	Enabled	Stopping	Fault_Stop	Faulted
<b>Switch Off</b>	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
<b>Switch on</b>	Transition to Startup	N/A	N/A	N/A	N/A	N/A	N/A	N/A
<b>Initialization Complete</b>	N/A	Transition to Not_Ready	N/A	N/A	N/A	N/A	N/A	N/A
<b>Main Power On</b>	N/A	N/A	Transition to Ready	N/A	N/A	N/A	N/A	N/A
<b>Run *</b>	N/A	N/A	N/A	Transition to Enabled	N/A	Transition to Enabled	N/A	N/A
<b>Stop *</b>	N/A	N/A	N/A	N/A	Transition to Stopping	N/A	N/A	N/A
<b>Stop Complete</b>	N/A	N/A	N/A	N/A	N/A	Transition to Ready	N/A	N/A
<b>Reset</b>	N/A	N/A	Transition to Startup	Transition to Startup	Transition to Startup	Transition to Startup	Transition to Startup	Transition to Startup
<b>Main Power Off</b>	N/A	N/A	N/A	Transition to Not_Ready	Transition to Faulted	Transition to Faulted	Transition to Faulted	N/A
<b>Fault Detected</b>	N/A	Transition to Faulted	Transition to Faulted	Transition to Faulted	Transition to Fault_Stop	Transition to Fault_Stop	N/A	N/A
<b>Fault_Stop Complete</b>	N/A	N/A	N/A	N/A	N/A	N/A	Transition to Faulted	N/A
<b>Fault Reset</b>	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.

	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.

<b>Run1</b>	<b>Run2</b>	<b>Trigger Event</b>	<b>Run Type</b>
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.

<b>Code Value [Hex]</b>	<b>Meaning</b>
<b>0000</b>	<b>No fault</b>
<b>1000</b>	<b>General Fault</b>
<b>2000</b>	<b>Current</b>
2100	Current, Device Input Side
2110	Short Circuit/Short to Earth
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

<b>Code Value [Hex]</b>	<b>Meaning</b>
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
3220	Undervoltage inside the Device
3221	Undervoltage No. 1
3222	Undervoltage No. 2
3230	Charging Error
3300	Output Voltage
3310	Output Overvoltage
3311	Output overvoltage in Phase U
3312	Output overvoltage in Phase V
3313	Output overvoltage 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

<b>Code Value [Hex]</b>	<b>Meaning</b>
4410	Excess Supply Temperature
4420	Inadequate Supply Temperature
<b>5000</b>	<b>Device Hardware</b>
5100	Power Supply
5110	Low Voltage Power Supply
5111	$\pm 15V$ 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
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

<b>Code Value [Hex]</b>	<b>Meaning</b>
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
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 Synchronisation
F003	Lifting Mechanism
F004	Open LOOP Control

## 5-30. AC/DC DRIVE OBJECT

Class Code: 2Ahex

This object models the functions specific to an AC or DC Drive. e.g. speed ramp, torque control etc.

### 5-30.1. AC/DC Drive 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. AC/DC Drive 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	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

Attribute ID	Need in implementation	Access Rule	Name	Data Type	Description of Attribute
9	Optional	Get	CurrentActual	INT	Actual motor phase current Units: 100ma / $2^{\text{CurrentScale}}$ where CurrentScale is attribute 23
10	Optional	Set/Get	CurrentLimit	INT	Motor phase current limit Units: 100ma/ $2^{\text{CurrentScale}}$ where CurrentScale is attribute 23
11	Optional	Get	TorqueActual	INT	Actual torque Units: Nm / $2^{\text{TorqueScale}}$ where TorqueScale is attribute 24
12	Optional	Set/Get	TorqueRef	INT	Torque reference Units: Nm / $2^{\text{TorqueScale}}$ where TorqueScale is attribute 24
13	Optional	Get	ProcessActual	INT	Actual process control value Units: % / $2^{\text{ProcessScale}}$ where ProcessScale is attribute 25
14	Optional	Set/Get	ProcessRef	INT	Process control reference set point Units: % / $2^{\text{ProcessScale}}$ where ProcessScale is attribute 25
15	Optional	Get	PowerActual	INT	Actual output power Units: Watts / $2^{\text{PowerScale}}$ where PowerScale is attribute 26
16	Optional	Get	InputVoltage	INT	Input Voltage Units: Volts / $2^{\text{VoltageScale}}$ where VoltageScale is attribute 27
17	Optional	Get	OutputVoltage	INT	Output Voltage Units: Volts / $2^{\text{VoltageScale}}$ where VoltageScale is attribute 27
18	Optional	Set/Get	AccelTime	UINT	Acceleration time Time from 0 to HighSpdLimit Units: ms / $2^{\text{TimeScale}}$ 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 / $2^{\text{TimeScale}}$ 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 / $2^{\text{SpeedScale}}$ where SpeedScale is attribute 22
21	Optional	Set/Get	HighSpdLimit	UINT	Maximum speed limit Units: RPM / $2^{\text{SpeedScale}}$ where SpeedScale is attribute 22
22	Optional*	Set/Get	SpeedScale	SINT	Speed scaling factor. Scaling is accomplished as follows: ScaledSpeed = RPM / $2^{\text{SpeedScale}}$ Range: -128 .. 127

Attribute ID	Need in implementation	Access Rule	Name	Data Type	Description of Attribute
23	Optional*	Set/Get	CurrentScale	SINT	Current scaling factor. Scaling is accomplished as follows: ScaledCurrent = A / 2 <sup>CurrentScale</sup> Range: -128 .. 127
24	Optional*	Set/Get	TorqueScale	SINT	Torque scaling factor. Scaling is accomplished as follows: ScaledTorque = Nm / 2 <sup>TorqueScale</sup> Range: -128 .. 127
25	Optional*	Set/Get	ProcessScale	SINT	Process scaling factor. Scaling is accomplished as follows: ScaledProcess = % / 2 <sup>ProcessScale</sup> Range: -128 .. 127
26	Optional*	Set/Get	PowerScale	SINT	Power scaling factor. Scaling is accomplished as follows: ScaledPower = W / 2 <sup>PowerScale</sup> Range: -128 .. 127
27	Optional*	Set/Get	VoltageScale	SINT	Voltage scaling factor. Scaling is accomplished as follows: ScaledVoltage = V / 2 <sup>VoltageScale</sup> Range: -128 .. 127
28	Optional*	Set/Get	TimeScale	SINT	Time scaling factor. Scaling is accomplished as follows: ScaledTime = ms / 2 <sup>TimeScale</sup> 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 <sup>CurrentScale</sup> 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 <sup>CurrentScale</sup> where CurrentScale is attribute 23
36	Optional	Set/Get	FieldMinCur	INT	Minimum Field Current for a DC Drive. Units: Amps / 2 <sup>CurrentScale</sup> where CurrentScale is attribute 23

\* See section 5-30.5.1. for scaling example.

The basic units used by the AC/DC Drive Object are:

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

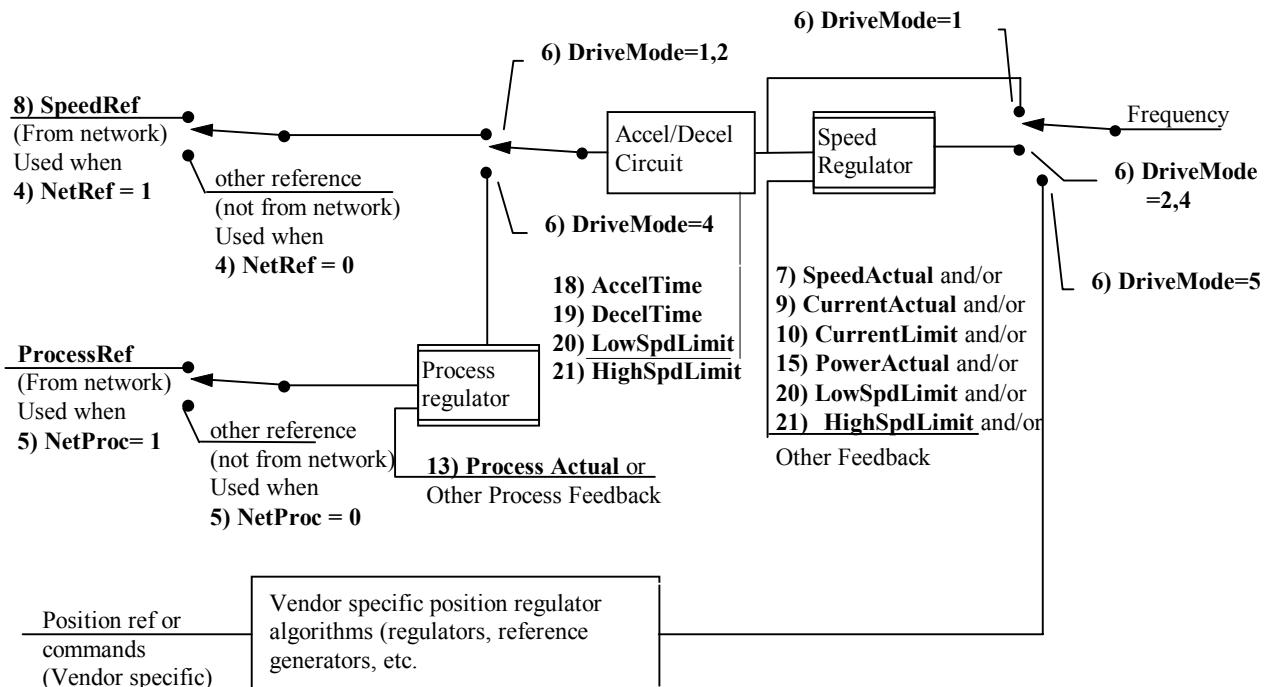
### 5-30.3. AC/DC Drive Common Services

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 Ready or NOT_READY states.
16 <sub>hex</sub>	n/a	Optional	Save	Saves all attribute values to a location accessible by the Restore service.

### 5-30.4. AC/DC Drive Object Specific Services

The AC/DC Drive object provides no object specific services.

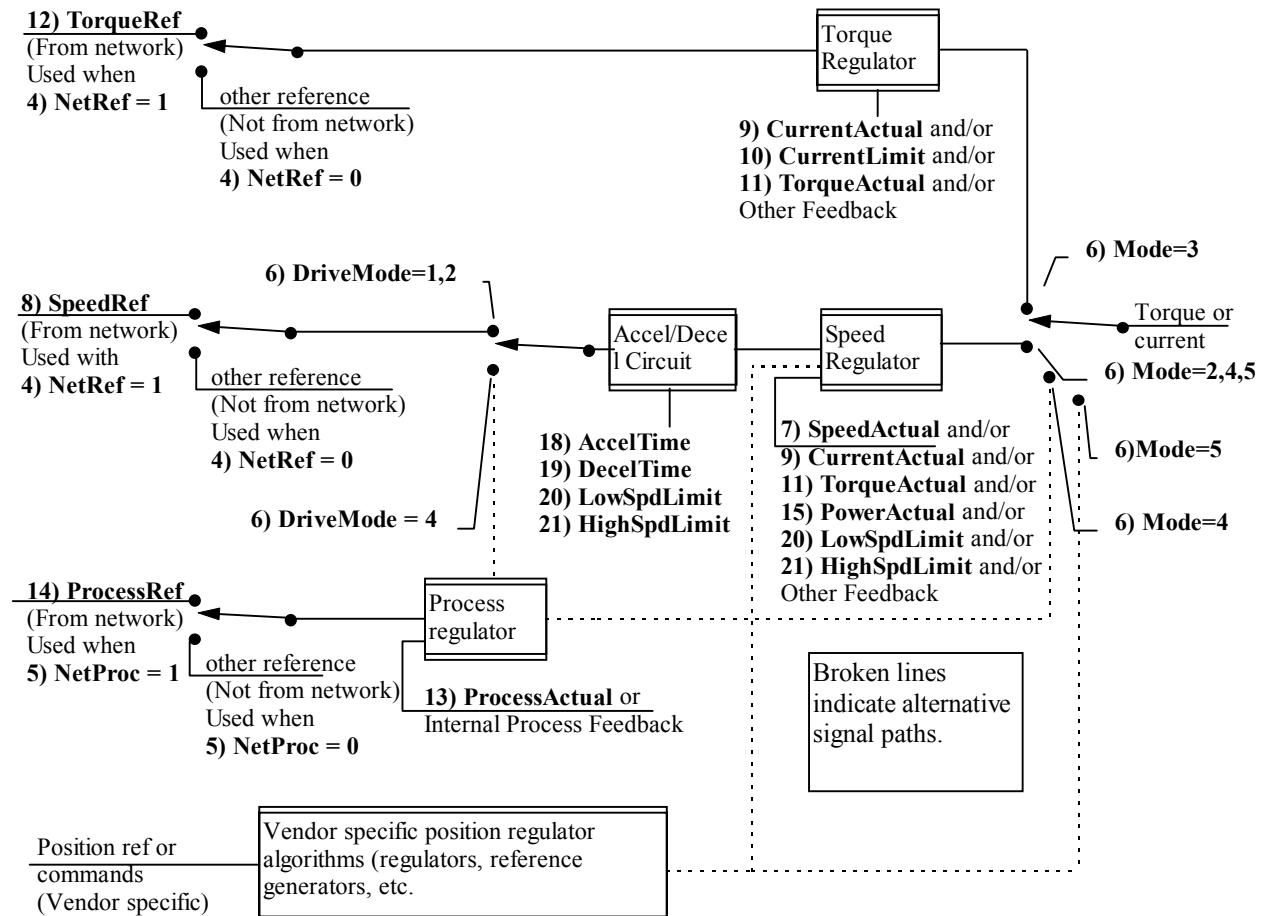
### 5-30.5. AC/DC Drive Object Behavior



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.

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.

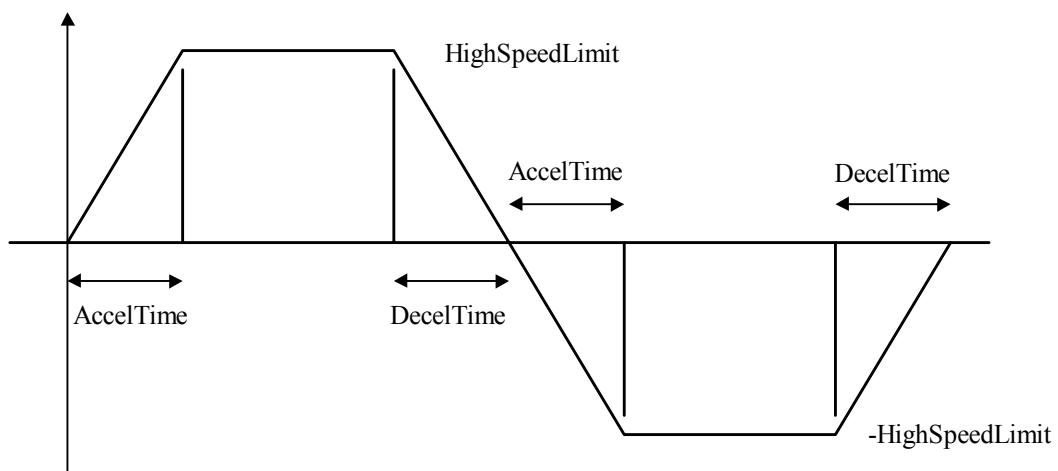
Each drive vendor can implement a process regulator and position regulator to feed either the speed regulator or the torque command directly. Broken lines indicate these alternative signal paths.



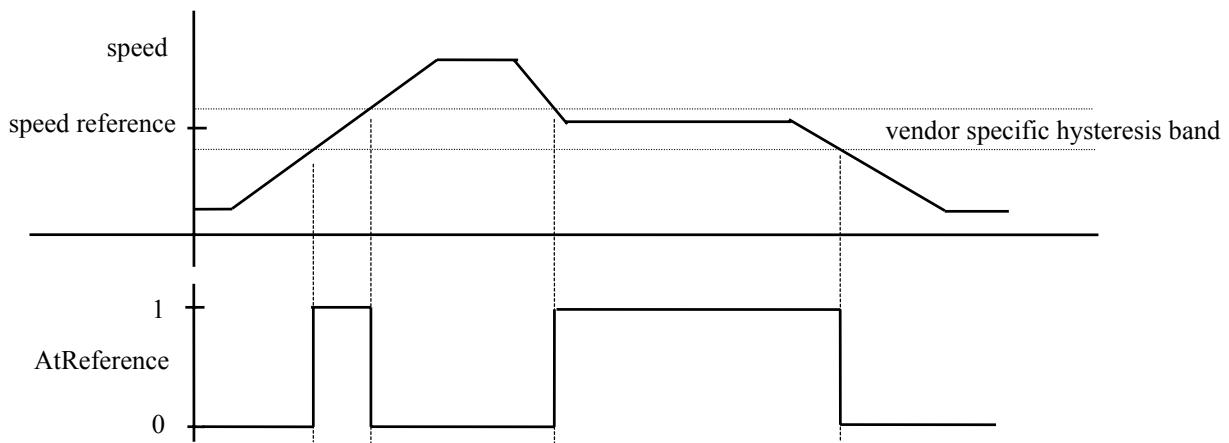
As illustrated below, **AccelTime** (attribute 18) is defined as the time in seconds it would take the drive to accelerate from zero to **HighSpdLimit** (attribute 21), the maximum speed at which the drive is allowed to operate.

Similarly, **DecelTime** (attribute 19) is the time to slow down from **HighSpdLimit** (attribute 21) to zero speed.

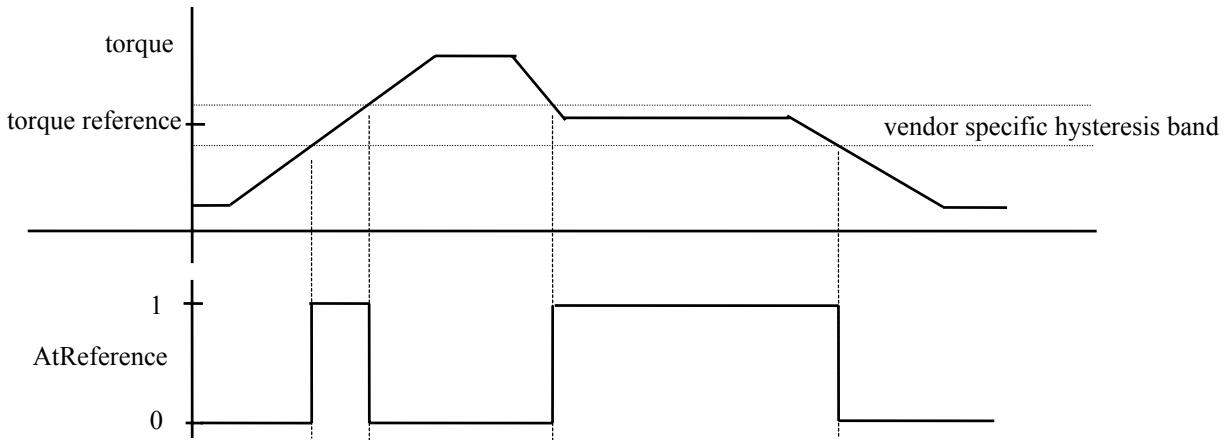
There will be no guarantee that the drive will operate at these rates in actual applications, since current limit may increase the acceleration time, and lack of sufficient braking capability may increase the stopping time.



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.



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.



### 5-30.5.1. 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

SpeedRef (AC/DC Drive Object, Attribute ID 8) = 4567

SpeedScale (AC/DC Drive Object, Attribute ID 22) = 3

$$\Rightarrow \text{Actual Commanded Speed} = \frac{\text{SpeedRef}}{2^{\text{SpeedScale}}} = \frac{4567}{2^3} = 570.875 \text{ RPM}$$

Input from Drive to bus:

Actual Drive Operating Speed = 789.5 RPM

SpeedScale (AC/DC Drive Object, Attribute ID 22) = 3

$$\Rightarrow \text{SpeedActual (AC/DC Drive Object, Attribute ID 7)} = \text{Actual Operating Speed} \times 2^{\text{SpeedScale}} \\ = 789.5 \times 2^3 = 6316.$$

In cases where the applicable scaling factor attribute is non-zero, the units are:

$$\frac{\text{engineering unit}}{2^{\text{Scale Factor Attribute}}}.$$

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:

$$\text{Frequency Command (Hz)} = \frac{\text{SpeedRef} \times \text{RatedFreq}}{2^{\text{SpeedScale}} \times \text{BaseSpeed}}$$

and an example implementation for the reported operating speed as a function of the output frequency may be:

$$\text{SpeedActual} = \text{Int} \left[ \frac{\text{Output Frequency (Hz)} \times \text{BaseSpeed} \times 2^{\text{SpeedScale}}}{\text{RatedFreq}} \right]$$

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: 2Bhex

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

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. Acknowledge Handler Object Class Services:

- Get\_Attribute\_Single
- Create
- Delete

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.

### 5-31.3. Acknowledge Handler Object Instance Attributes:

Attr ID	Need in Implementation	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.
- 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. Acknowledge Handler Object Instance Services:

The Acknowledge Handler Object Instance supports the following Common Services:

- Get\_Attribute\_Single
- Set\_Attribute\_Single
- Delete

Need in Implementation	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.

### 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

Need in Implementation	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).

#### 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.
11. The number of retries is (optionally) configurable within the Acknowledge Handler object.

### 5-31.6.1. Change of State Examples

#### 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.

### 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.

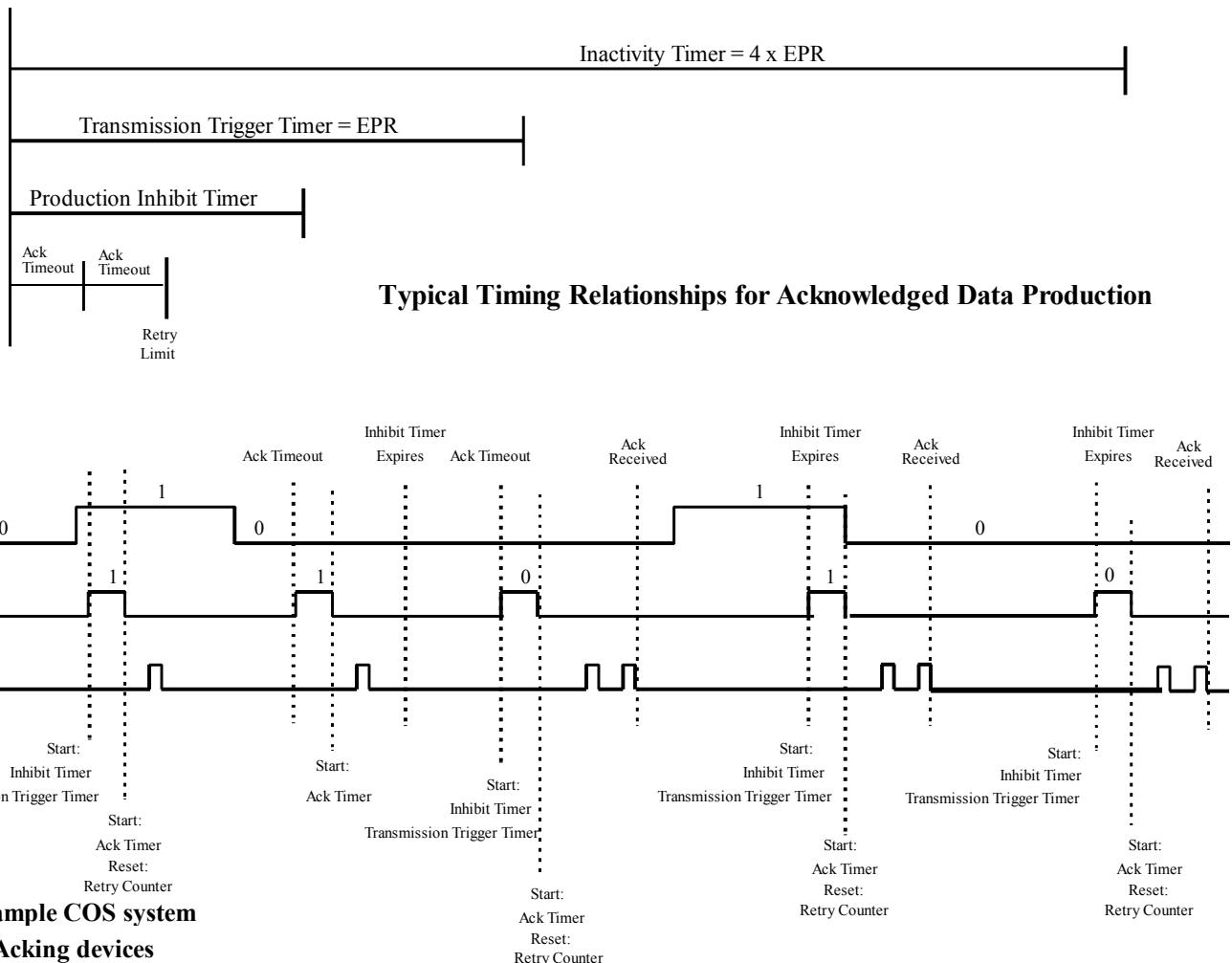
*For each consumer:*

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.

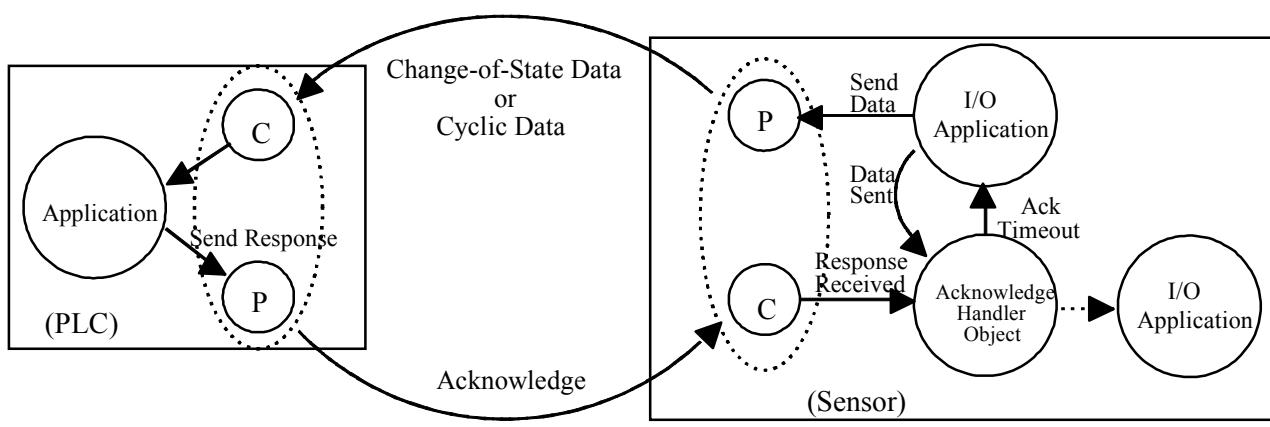
Once the Connection Object(s) is (are) configured, an Apply Attributes service is sent to each configured Connection Object to transition the connection(s) to the Established state. During the apply, the connection object is 'delivered' to both the consuming and producing application objects for validation 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.

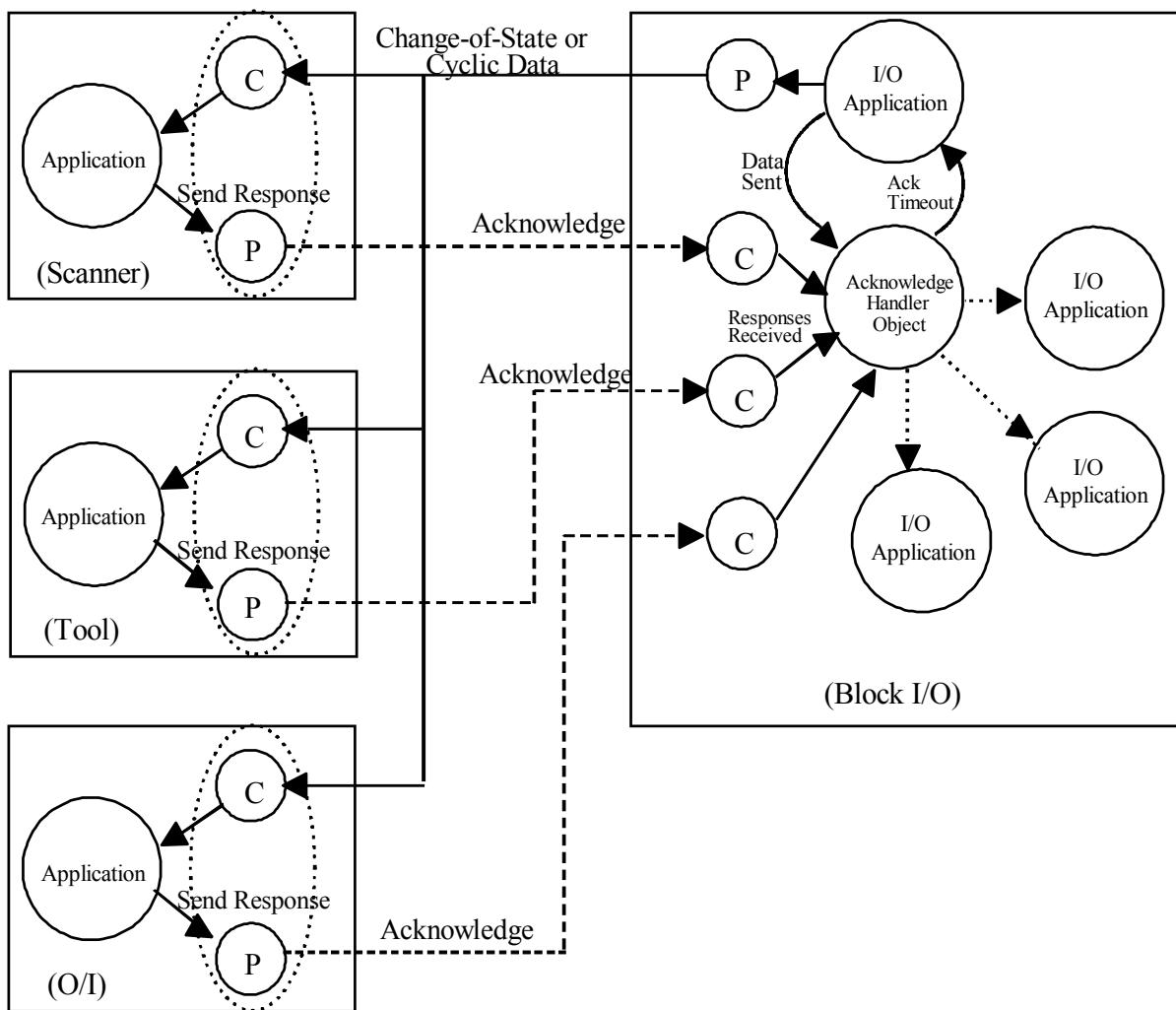
### 5-31.6.2. Use of Timers with Acknowledged Data Production

1. The following rules must be observed when sending acknowledged data.
2. New data not sent while the Inhibit Timer is active (running).
3. 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.
4. Retry of old data occurs at Ack Timeout if new data is not available or the Inhibit Timer is active.
5. 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.
6. A retry of old data starts the Ack Timer.



The following diagrams illustrate the message flow in a Change of State connection for both single and multi-consumer configurations.





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.

**SEM for Producing I/O Application Object**

<b>State Event</b>	<b>Not Running</b>	<b>Running</b>	<b>Running with Acknowledgement</b>	<b>Prohibited</b>
<b>Change of State Detected</b>	Ignore Event	Inform Link Producer to Send Data If Inhibit Time configured Start Inhibit Timer Transition to Prohibited	Inform Link Producer to Send Data Send Data_Sent 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_Received 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 with 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 Acknowledgement 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.*

SEM for Acknowledge Handler Object

State Event \ Non-Existent	Non-Existent	Inactive	Active
Receive Acknowledge	Not Applicable	Ignore Event	<p>Clear Ack Flag</p> <p>Forward any data to application object</p> <p>If all Acknowledges received</p> <ul style="list-style-type: none"> <li>Clear Ack Timer and Retry Counter</li> <li>Send Acknowledge_Received event message to producing application object</li> </ul>
Acknowledge Timer Expires	Not Applicable	Ignore Event	Send Ack_Timeout event message to producing application object
Data_Sent	Not Applicable	Ignore Event	<p>Set Ack Required flag</p> <p>Set Ack Timer</p> <p>Clear Retry Counter</p>
Data_Resent	Not Applicable	Ignore Event	<p>Set Ack Required Flag and Ack Timer</p> <p>If Retry_Limit &lt;&gt; 0</p> <p>Increment Retry Counter</p> <p>If Retry Counter = Retry_Limit</p> <p>Send Retry_Limit_Reached event message to producing application object</p>
Delete	Not Applicable	Transition to Non-Existent	Transition to Non-Existent
Create	Transition to Inactive	Not Applicable	Not Applicable
Apply_Attributes	Not Applicable	<p>Verify new connection can be added to list</p> <p>Pass this message to the consumed application object, if one is configured for this connection</p>	<p>Verify new connection can be added to list</p> <p>Pass this message to the consumed application object, if one is configured for this connection</p>
Connection Transition to Established	Not Applicable	<p>Add this connection instance to the connection list (or internally flag as 'Acking')</p> <p>Pass this message to the consumed application object, if one is configured for this connection</p> <p>Send Acknowledge_Active event message to producing application</p> <p>Transition to Active</p>	<p>Add this connection instance to the connection list</p> <p>Pass this message to the consumed application object, if one is configured for this connection</p>
Inactivity/Watchdog Timer Expires	Not Applicable	Not Applicable	<p>Internally flag this connection as 'Not Acking'. An acknowledge will no longer be monitored for this connection, however it remains in the connection list.</p> <p>Pass this event to the consumed application object, if one is configured for this connection</p> <p>If no 'Acking' connections in list send Acknowledge_Inactive event to producing application and transition to Inactive.</p>
Connection Deleted	Not Applicable	Ignore Event	<p>Remove this connection instance from the connection list</p> <p>Pass this event to the consumed application object, if one is configured for this connection</p> <p>If no 'Acking' connections in list send Acknowledge_Inactive event to producing application and transition to Inactive.</p>

## 5-32. OVERLOAD OBJECT

Class Code: 2Chex

This object models all the functions specific to an AC motor overload protection device.

### 5-32.1. Overload 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. Overload 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 CurrentScale Units: 100ma / 2 where CurrentScale is attribute 12
9	Optional	Get	CurrentL2	INT	Actual motor phase current L2 CurrentScale Units: 100ma / 2 where CurrentScale is attribute 12
10	Optional	Get	CurrentL3	INT	Actual motor phase current L3 CurrentScale Units: 100ma / 2 where CurrentScale is attribute 12
11	Optional	Get	GroundCurrent	INT	Ground Current CurrentScale Units: 100ma / 2 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

$$\Rightarrow \text{Actual FLC setting used by device} = \text{TripFLCSet} = 23456 \div 2^8 \\ = 91.625 \text{ mA} = 9.1625 \text{ amps}$$

Input (from drive to bus):

Actual FLC setting used by device = 78.95 amps = 789.5 mA

CurrentScale (Overload Object, Attribute ID 12) = 3

$$\Rightarrow \text{TripFLCSet(Overload Object, Attribute ID 3)} = 789.5 \times 2^8 = 6316$$

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. Overload 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.

### 5-32.4. Overload Object Specific Services

The Overload object provides no object specific services.

### 5-32.5. Overload Object Behavior

The Overload object behaviour can be changed by setting attributes, overload current setting and Trip Class setting will effect the trip curve of the device.

### 5-32.6. Fault and Warning codes

This table lists the fault and warning codes used by AC Motor Starter Device Profiles (Overload and Softstarter)

Code Value	Meaning
0	NO FAULT
10	TEST
11	FORCE TRIP
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
40	CONTROL VOLTAGE
41	CONTROL UNDERTVOLTAGE
42	CONTROL OVERVOLTAGE
43	FREQUENCY
50	MAIN VOLTAGE
51	UNDERTVOLTAGE
52	OVERVOLTATGE
53	IMBALANCE
54	PHASE REVERSAL
55	FREQUENCY
60	HARDWARE
61	ILLEGAL FLC SETTING
62	MEMORY FAULT
63	HARDWARE LINK FAULT
64	NO DEVICE POWER
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

\* Note Fault Codes 101 to 255 are Vendor Specific Codes

## 5-33. SOFTSTART OBJECT

Class Code: 2Dhex

This object models all the functions specific to a Soft Start Motor Starter.

### 5-33.1. Softstart 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. Softstart 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

Attribute ID	Need in implementation	Access Rule	Name	Data Type	Description of Attribute
12	Optional	Get/Set	KickStart	BOOL	0 =disabled 1 = enabled fixed time
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. Softstart 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.

### 5-33.4. Softstart Object Specific Services

The Softstart object provides no object specific services.

### 5-33.5. Softstart Object 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).

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 7) = 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 Ouput 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

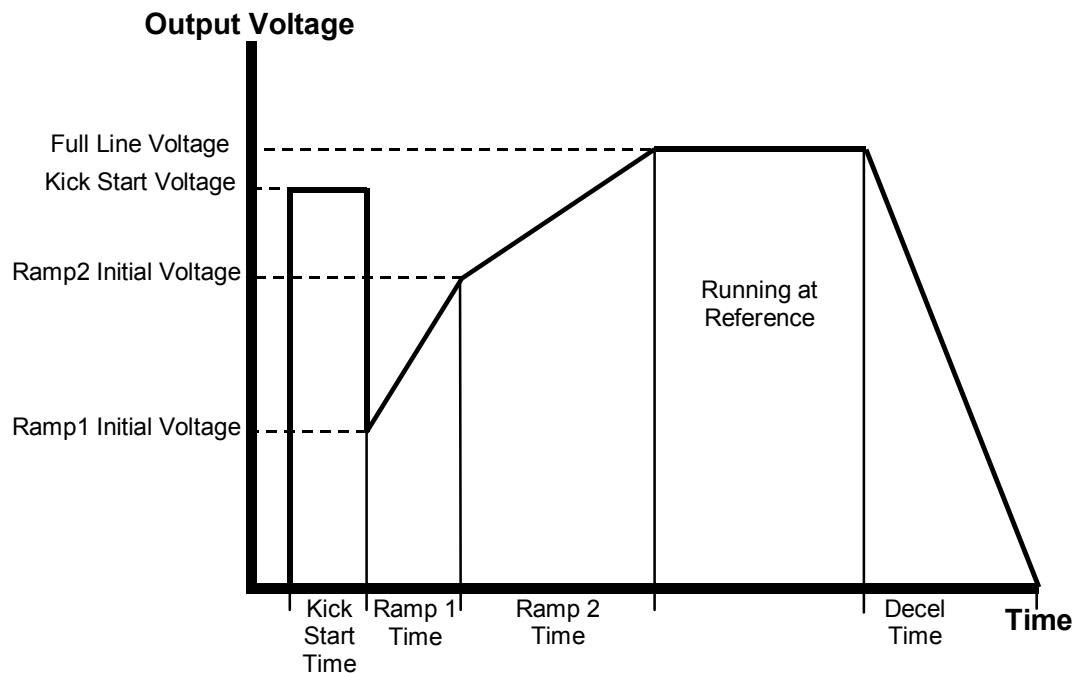
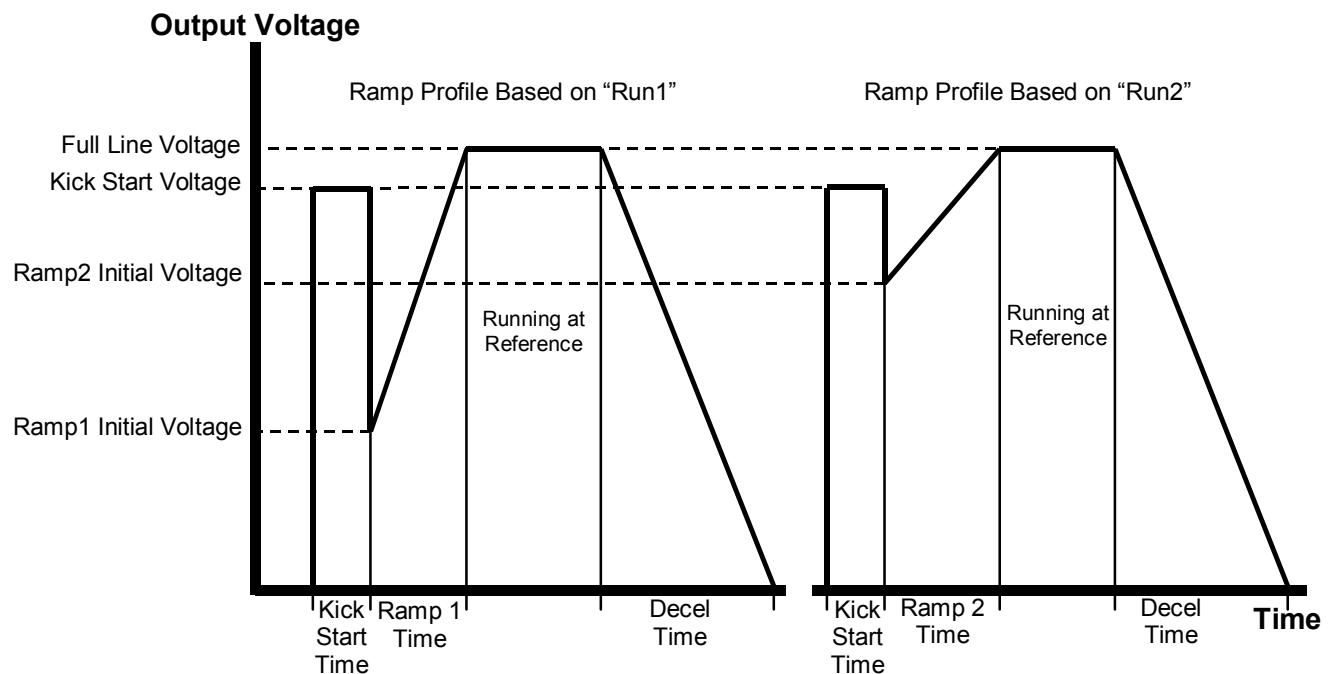
Rotation (Attribute 11)

Sets the main voltage, phase rotation sequence for Phases A,B & C.

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.

Soft Starter Output Voltage Behavior Description

***Example: Dual Contiguous Ramp Soft Start With Ramp Down Stop Mode******Example: Dual Independent Ramp Soft Start With Ramp Down Stop Mode***

### 5-33.6. Fault and Warning codes

This table lists the fault and warning codes used by AC Motor Starter Device Profiles (Overload and Softstarter)

Code Value	Meaning
0	NO FAULT
10	TEST
11	FORCE TRIP
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
40	CONTROL VOLTAGE
41	CONTROL UNDERTOLAGE
42	CONTROL OVERVOLTAGE
43	FREQUENCY
50	MAIN VOLTAGE
51	UNDERTOLAGE
52	OVERVOLTATGE
53	IMBALANCE
54	PHASE REVERSAL
55	FREQUENCY
60	HARDWARE
61	ILLEGAL FLC SETTING
62	MEMORY FAULT
63	HARDWARE LINK FAULT
64	NO DEVICE POWER
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

\* Note Fault Codes 101 to 255 are Vendor Specific Codes

## 5-34. SELECTION OBJECT

Class Code: 2Ehex

The Selection Object manages selection and distribution of I/O messages from the Connection Object to internal Application Objects.

### 5-34.1. 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.					
8	Reserved for CIP					
9	Optional	Get	Max number of instances	UINT	Indicates maximum number of instances supported.	Range 0 – 65535.

### 5-34.2. 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

Attr ID	Need in Implementation	Access Rule	Attribute Name	Data Type	Attribute Description	Semantics of Values
1	Required	Get	State	USINT	State of the object.	See attribute description.
2	Required	Get	Max_destinations	UINT	Indicates maximum number of destination indexes supported.	Range 0-65535.
3	Required	Get	Number_of_destinations	UINT	Number of entries of destination_list.	Range 0-65535.
4	Required	Set	Destination_list	Array of Path	Path ::= SEQUENCE OF { [ USINT pathlength; Array of USINT] }	Path length, Path. See Appendix C for encoding.
5	Required	Get	Max_Sources	UINT	Indicates maximum number of source indexes supported.	Range 0-65535.
6	Required	Get	Number_of_sources	UINT	Number of entries of source_list.	Range 0-65535.
7	Required	Set	Source_list	Array of UINT	Array of consuming I/O Connection Instance IDs.	Connection Instance IDs.
8	Optional	Get	Source_used	UINT	Index into	Range 0-65535

Attr ID	Need in Implementation	Access Rule	Attribute Name	Data Type	Attribute Description	Semantics of Values
					source_list array indicating which source is currently being used.	where one (1) is first entry in source_list.
9	Optional	Get	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	Algorithm_Type	USINT	The selection algorithm type.	If not supported, algorithm_type =0.
11	Conditional	Get/Set	Detection_Count	USINT	Required when algorithm type is supported.	Range 2-255. Default value is 4.
12	Conditional	Get/Set	Selection_Period	UINT	Required when algorithm type is supported.	Range 1-65535 (ms).

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.

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.

2. **max\_destinations** - Maximum number of destination paths managed by the Selection Object.
3. **number\_of\_destinations** - Number of destination paths managed by the Selection Object. Default value is zero(0).
4. **destination\_list** - This attribute is an array of paths to objects which receive messages from the Selection Object. See Volume I, Appendix I. All members in the list must be unique.
5. **max\_sources** - Maximum number of source paths managed by the Selection Object.
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.
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.
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.
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.
10. **algorithm\_type** - Defines the algorithm types. All four algorithm types are required if this attribute is supported. If this attribute is not supported, then all messages pass through. See table below.

### Values assigned to the algorithm\_type attribute

Value	State Name	Description
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.
4-63	Reserved	Reserved for future use (Open).
64-C7	Vendor Specific	
C8 -FF	Reserved	Reserved for future use.

11. **detection\_count** - The parameter used by algorithms which require a counter.
12. **selection\_period** - The parameter used by algorithms which require a time interval.

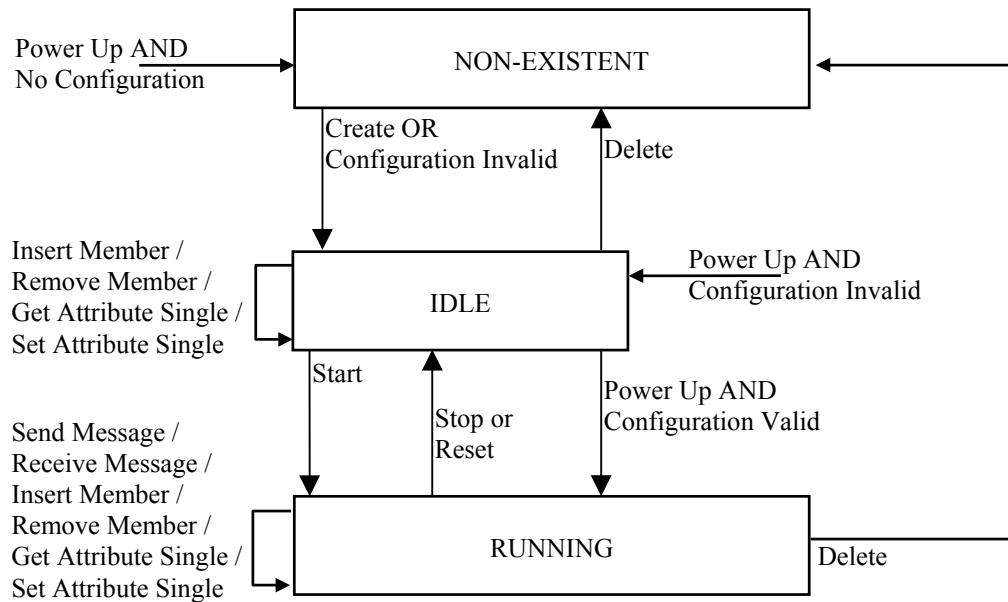
### 5-34.4. Instance Services

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.

### 5-34.5. Instance Behavior

The behavior of the Selection Object, while in the RUNNING state, is dependent upon the algorithm\_type attribute.

**The Selection Object State Transition Diagram**



**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

Event	Non-Existent	Idle	Running
			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.

### Selection Object Attribute Access

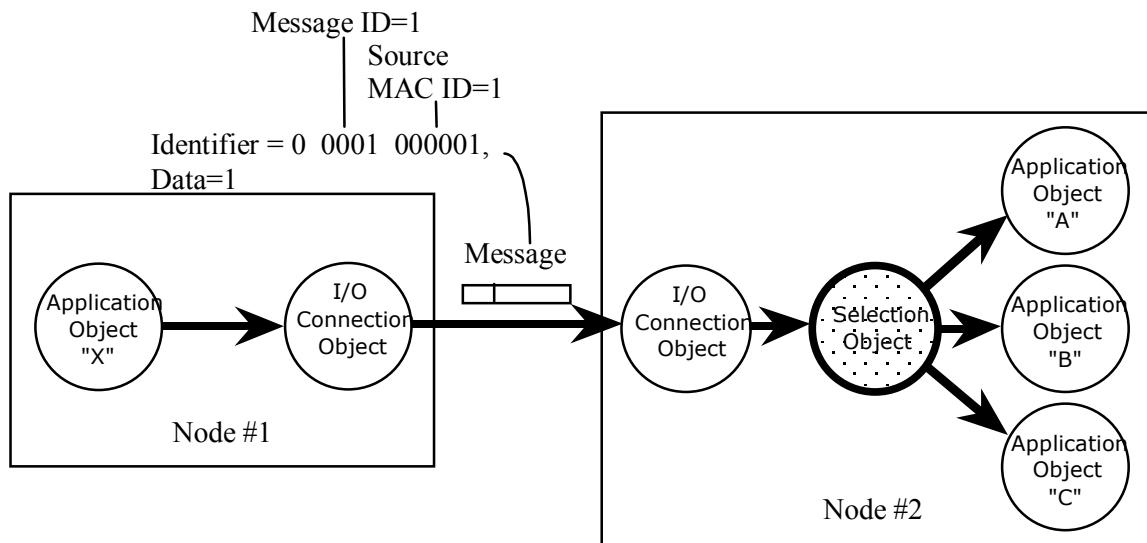
Attribute	Non-Existent	Idle	Running
State	Not available	Get only	Get only
max_destinations	Not available	Get only	Get only
number_of_destinations	Not available	Get only	Get only
destination_list	Not available	Get/Set/Insert/Remove	Get/Set/Insert/Remove
max_sources	Not available	Get only	Get only
number_of_sources	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

#### 5-34.5.1. 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).

### I/O Message Distribution

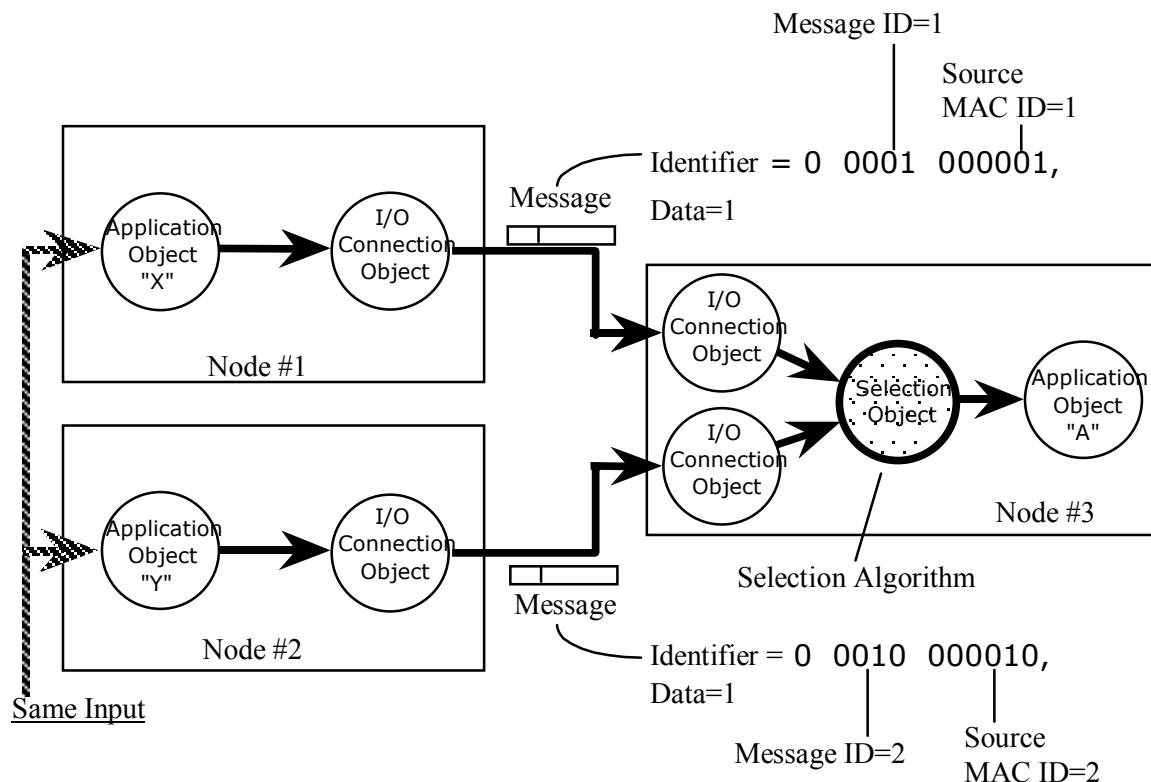


#### 5-34.5.2. Message Selection

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.

## Message Selection Example



The following algorithms are currently defined for message selection.

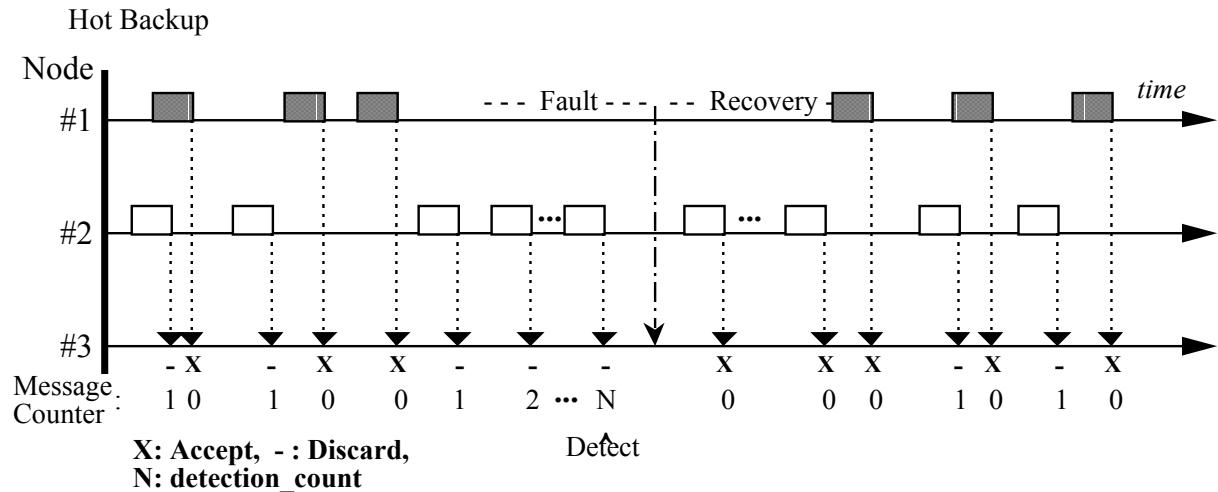
- Pass Through
- Hot Backup
- First Arrival

#### 5-34.5.3. Pass Through

No algorithm. Forward all arriving messages to destination(s).

#### 5-34.5.4. Hot Backup

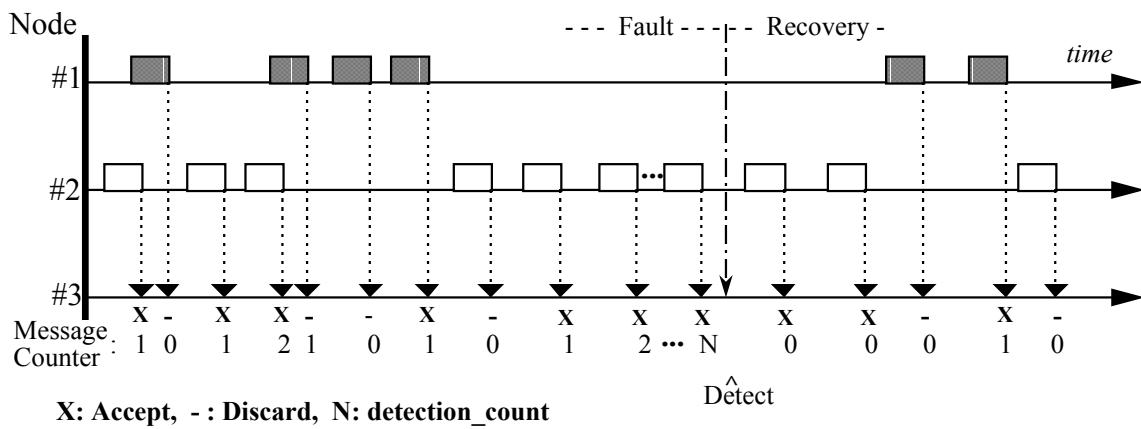
This algorithm is shown in the Figure labeled Hot Backup. Node #1 is higher priority than Node #2. Node #3 normally accepts messages from Node #1. Node failure is detected in the following manner. If the Selection Object detects that Node #1 failed, it accepts messages from Node #2 and forwards them 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.



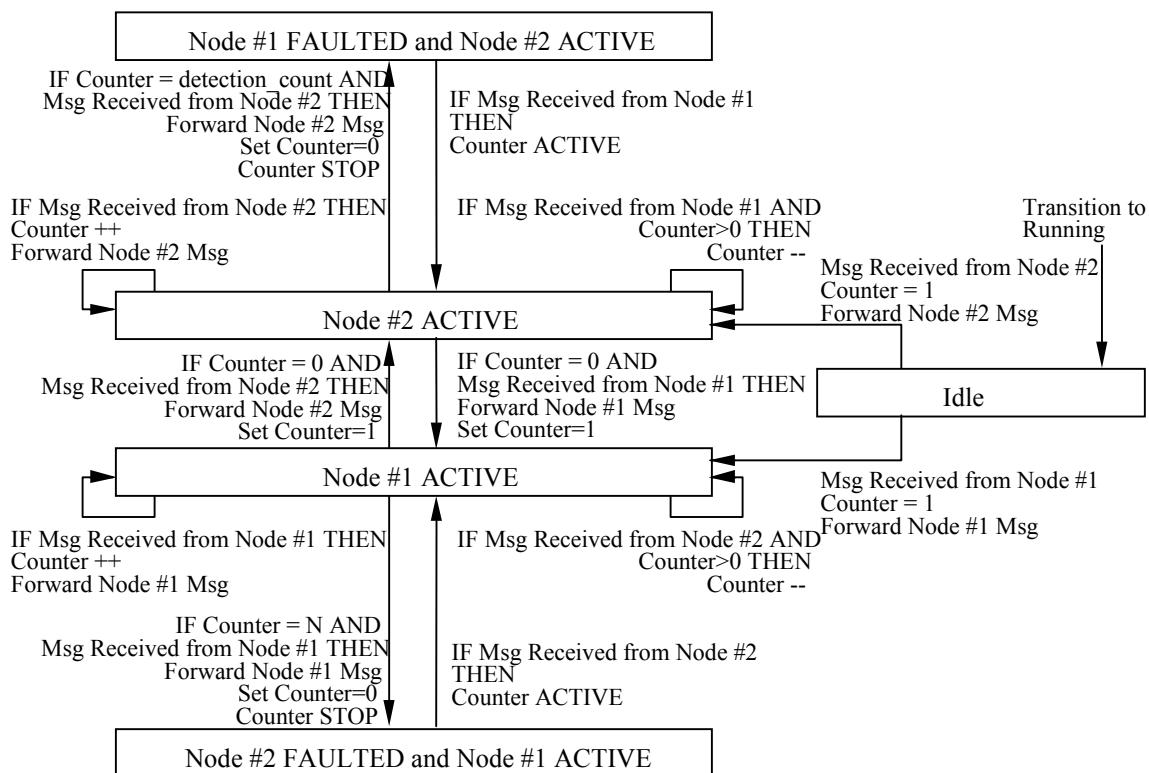
### 5-34.5.5. First Arrival

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.

First Arrival



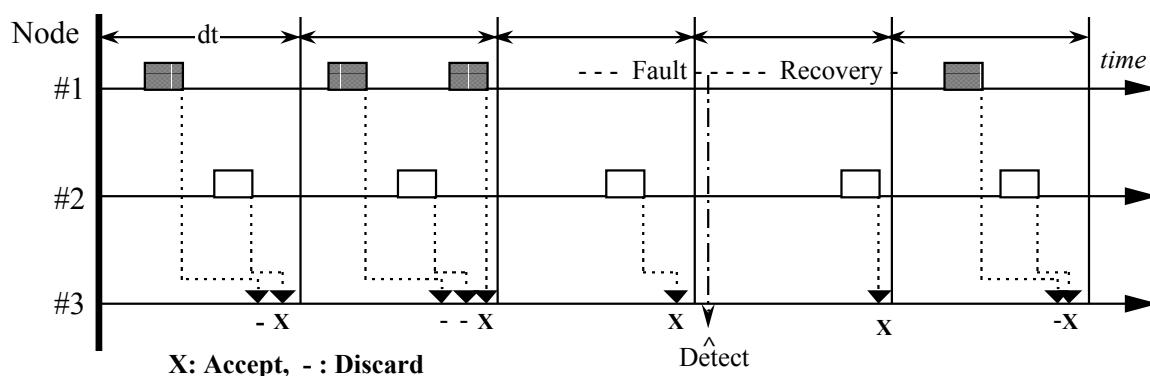
## Behavior of First Arrival in “Running State”



## 5-34.5.6. Last Arrival

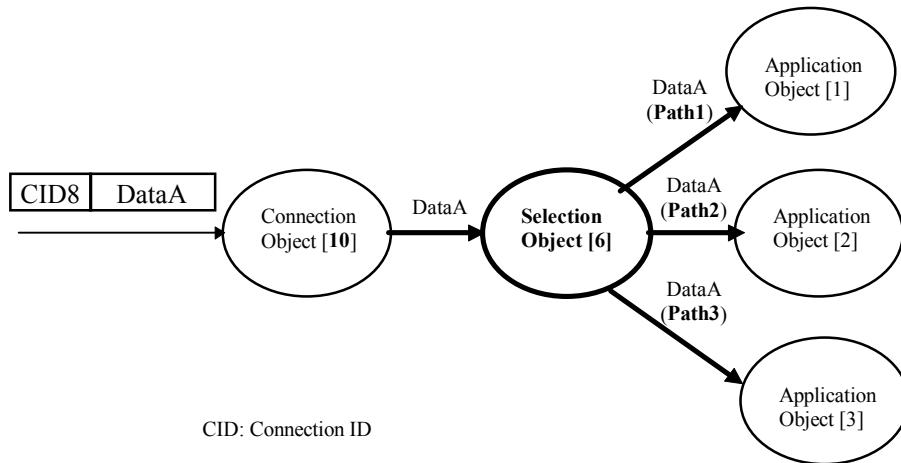
This algorithm is shown in the Figure labeled Last Arrival. The Selection Object starts a timer when transmits to running. It forwards the last message received when the timer expires, and immediately restarts the timer. If no message is received AND the timer expires, the Selection Object does not forward 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.

Last Arrival



### 5-34.5.7. Examples

#### I/O Message Distribution



Attribute Number	Attribute Name	Attribute Value
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

#### I/O Message Selection

Attribute Number	Attribute Name	Attribute Value
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

## 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\*, 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.

\* Semiconductor Equipment and Materials International, Mountain View CA, Standard E54: *Sensor/Actuator Network Common Device Model*.

### 5-35.1. S-Device Supervisor 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.

### 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. S-Device Supervisor 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.

Attr ID	Need in implementation	Access Rule	NV	Name	Data Type	Description of Attribute
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
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

<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>
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	V	<b>Exception Detail Alarm</b>	STRUCT of:	A Structure of three Structures containing a bit mapped representation of the alarm detail
				<b>Common Exception Detail</b>	STRUCT of:	
				Size	USINT	Number of Common Detail Bytes
				Detail	ARRAY of:	See “Semantics” section
				Detail n	BYTE	See “Semantics” section
				<b>Device Exception Detail</b>	STRUCT of:	
				Size	USINT	Number of Device Detail Bytes
				Detail	ARRAY of:	See Device Profile
				Detail n	BYTE	See Device Profile
				<b>Manufacturer Exception Detail</b>	STRUCT of:	
14	Conditional based on Exception Status Bit 7  Conditional based on Size  Conditional based on Size	Get	V	<b>Exception Detail Warning</b>	STRUCT of:	A Structure of three Structures containing a bit mapped representation of the warning detail
				<b>Common Exception Detail</b>	STRUCT of:	
				Size	USINT	Number of Common Detail Bytes
				Detail	ARRAY of:	See “Semantics” section
				Detail n	BYTE	See “Semantics” section
				<b>Device Exception Detail</b>	STRUCT of:	
				Size	USINT	Number of Device Detail Bytes
				Detail	ARRAY of:	See Device Profile
				Detail n	BYTE	See Device Profile
				<b>Manufacturer Exception Detail</b>	STRUCT of:	

Attr ID	Need in implementation	Access Rule	NV	Name	Data Type	Description of Attribute
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
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.
97-98	Reserved by CIP for Future Use					
99	Conditional **	Get	NV	Subclass	UINT	Identifies a subset of additional instance attributes, services and behaviors.

\* 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.

## 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.

## Semantics:

### 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.

### 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.

### 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.

### 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.

### 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.

### 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:

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

### 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.

	Exception Status Bit Map, Bit 7 set to 0	Exception Status Bit Map, Bit 7 set to 1
Bit	Function	Function
0	<b>Device Specific definition (See Device Profile)</b>	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.

### 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.

#### 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.

**Common Exception Detail Attribute Values**

	Common Exception Detail [0]*	Common Exception Detail [1]*
	Internal diagnostic exception	power supply overcurrent
	Microprocessor exception	reserved power supply
	EPROM exception	power supply output voltage
	EEPROM exception	power supply input voltage
	RAM exception	scheduled maintenance due
	Reserved by CIP	notify manufacturer

	Common Exception Detail [0]*	Common Exception Detail [1]*
	Internal real-time exception	reset exception
	Reserved by CIP	reserved by CIP

\* Note that if a device does not support an exception detail, the corresponding bit is never set

#### 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.

#### 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.

#### 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	—	—	—	—	—	—	—	—

Data Component	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Manufacturer Exception Detail Size	x	x	X	x	x	x	x	x
Manufacturer Exception Detail 0	—	—	—	—	—	—	—	—
Manufacturer Exception Detail n	—	—	—	—	—	—	—	—

#### 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).

#### 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.

#### 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.

#### 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. S-Device Supervisor 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

### 5-35.4. S-Device Supervisor 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

#### DS Object Service Parameter Dictionary

Parameter	Form	Description of Service
TestID	USINT	Type and possibly detail of diagnostic test to be performed

**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.

**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.

**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.

TestID Parameter

The following values are defined for the TestID parameter for the Perform\_Diagnostics Service Request:

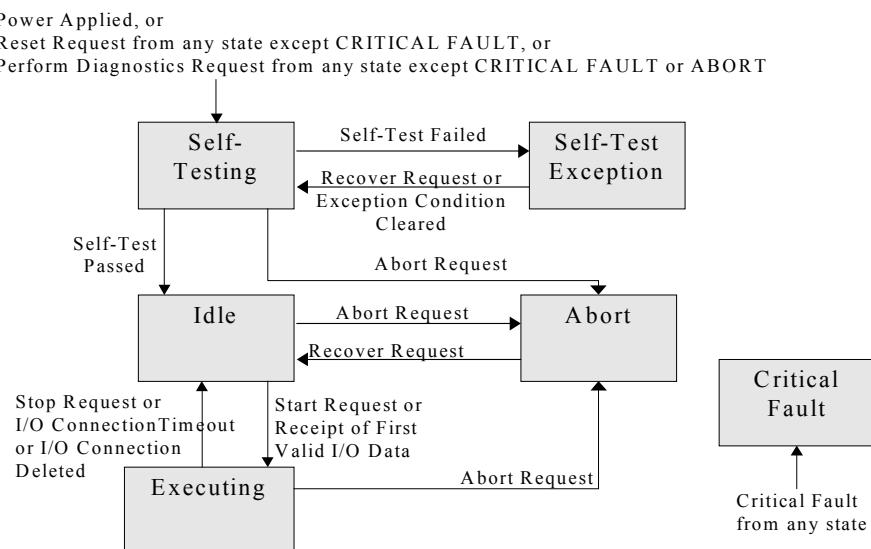
Attribute Value	State
0	Standard
1-63	Reserved
64-127	Device Specific (defined in Device Profile)
128-255	Manufacturer Specific (defined by manufacturer)

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.

## 5-35.5. S-Device Supervisor Behavior

### 5-35.5.1. S-Device Supervisor Object States

**Figure 5-35.1 DS Object Instance Behavior**



**Table 5-35.2 DS Instance Behavior State Description**

State	Description
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.

### 5-35.5.2. S-Device Supervisor State Event Matrix

**Table 5-35.3 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

Event	State					
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
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.

### 5-35.5.3. 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 5-35.4. 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
Deactivated	Stop Request *
Activated	Start Request *
Major Recoverable Fault	Abort Request *
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 5-35.5. 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
Reset Request	Deactivated
Start Request	Activated
Stop Request	Deactivated
Abort Request	Major Recoverable Fault

S-Device Supervisor Object Event	Identity Object Affected Event
Recover Request	Fault Corrected
Perform Diagnostics Request	No effect.*

\* The request to perform diagnostics has no effect on the Identity Object. However, the results of the tests may.

#### 5-35.5.4. 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.

## 5-36. 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

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.					
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.

### 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.

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 [default] = INT
4	Optional	See Semantics	NV	Data Units	ENGUNITS	Determines the Units context of <i>Value</i> and all related attributes.	see Semantics section [default] = Counts
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.	The value of attribute <i>Value</i> corresponding to the Full Scale calibrated measurement of the sensor. [default] = maximum allowable value for the <i>Data Type</i>
11	Optional	Get	NV	Offset-A Data Type	USINT	Determines the Data Type of attribute <i>Offset-A</i>	see Semantics section [default] = INT
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]

Attr ID	Need in Implementation	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
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 [default] = REAL
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 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	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.

Attr ID	Need in Implementation	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
23	Optional	Set	NV	Warning Hysteresis	INT or specified by <i>Data Type</i> if supported	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	Warning 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
25	Optional	Set	NV	Safe State	USINT	Specifies the behavior for the <i>Value</i> for states other than Execute	see Semantics section [default] = 0
26	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 [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 <i>Data Type</i> if supported	Specifies the highest valid <i>Value</i>	The value above which attribute <i>Reading Valid</i> is set to invalid. [default] = maximum allowable value for the <i>Data Type</i>

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 <u>and/or</u> <i>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	Gas Calibration Object Instance	UINT	Indicates which Gas 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
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 6-65535=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.

## 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.

## Semantics:

### 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.

#### Data Units

Specifies the context of *Value* and related attributes (such as, offset and trip points) for this object instance. See Appendix K 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*.

#### 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:

$$Value = Gain \bullet ( \text{Sensor Reading} + Offset-A ) + Offset-B$$

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.

#### Status

A bit mapped byte which indicates the Alarm and Warning Exception status of the object instance. The following definition applies:

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

#### 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.

### 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:

Attribute Value	State
0	Zero
1	Full Scale
2	Hold Last Value
3	Use Safe Value
4-50	Reserved
51-99	Device Specific
100-255	Vendor Specific

### 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.

### 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).

### 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).

### 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.

### Gas Calibration Object Instance

This attribute is used to select an instance of the S-Gas Calibration object. The selected S-Gas Calibration object instance provides the data with which an S-Analog Sensor object instance enacts the appropriate calibration algorithm for a given gas type.

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 S-Gas Calibration object.

**Conditionally Required:** If a device profile specifies an S-Gas Calibration object relationship for an S-Analog Sensor object instance, then this attribute is required. See the S-Gas Calibration object definition for more information.

### Produce Trigger Delta Type

The *Produce Trigger Delta Type* attribute specifies the interpretation of the *Produce Trigger Delta* attribute. The following values are defined:

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. Common Services

The S-Analog Sensor Object provides the following 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.

### 5-36.4.1. Object-Specific Services

The S-Analog Sensor Object provides the following 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.

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.

#### 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.

#### 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.

### 5-36.5. 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.

#### 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.

### 5-36.6. 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.

## CLASS ATTRIBUTES

The following Class Attributes are specified for this object subclass.

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 <i>Value</i> (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]

### Semantics:

#### 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.

#### 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.

#### 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.

## SERVICES

There are no additions or restrictions to the Object Services for this object subclass.

## 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.

### 5-36.7. 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.

## INSTANCE ATTRIBUTES

The following Instance Attributes are specified for this object subclass.

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

## SERVICES

There are no additions or restrictions to the Object Services for this object subclass.

## BEHAVIOR

### 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.

## 5-36.8. S-Analog Sensor Object Instance Subclass 02 (Heat Transfer Vacuum Gauge)

### OVERVIEW

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.

### 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.

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

### Semantics:

#### Cable Length

This attribute specifies the cable length of the transducer cable (cable between the sensor [filament] and its electronic).

#### Sensor Temperature

Ambient temperature of the sensor housing. This value could be used for temperature compensation.

#### 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.

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	Reserved	Reserved	Reserved	Reserved	Reserved	Sensor High Temperature Warning	Electronics Warning	Reserved

### 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.

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	Reserved	Reserved	Reserved	Reserved	Reserved	Over Temperature of Electronics	Electronics Failure	Reserved

### Status Extension

8 Bits providing the current sensor alarm state of the object.

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

## SERVICES

There are no additions or restrictions to the Object Services for this object subclass.

## BEHAVIOR

There are no additions or restrictions to the Object Behavior for this object subclass.

### 5-36.9. S-Analog Sensor Object Instance Subclass 03 (Diaphragm Gauge)

#### OVERVIEW

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.

#### 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.

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

## Semantics:

### 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 Pressure/Vacuum Gauge device profile. The following table provides definitions for each Sensor Warning bit. Reserved bits must be set to zero.

Data Component	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Sensor Warning Byte 0	Reserved	Not At Temperature						
Sensor Warning Byte 1	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Electronics Warning	Reserved

### 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 Pressure/Vacuum Gauge device profile. The following table provides definitions for each Sensor Alarm bit. Reserved bits must be set to zero.

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	Reserved	Reserved	Reserved	Reserved	Reserved	Over Temperature of Electronics	Electronics Failure	Reserved

### Status Extension

8 Bits providing the current sensor alarm state of the object.

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

## SERVICES

There are no additions or restrictions to the Object Services for this object subclass.

## 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.

## 5-36.10 S-ANALOG SENSOR OBJECT INSTANCE SUBCLASS 04 (COLD CATHODE ION Gauge)

### OVERVIEW

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.

### INSTANCE ATTRIBUTES

Certain minimal implementations may support any optional "Set" attributes as "Get" only and still be compliant with this object specification.

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

Semantics:

#### 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.

#### Sensitivity

The conversion factor used to convert the ratio of gauge currents to pressure. This conversion is manufacturer specific.

### 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 Pressure/Vacuum Gauge device profile. The following table provides definitions for each Sensor Warning bit. Reserved bits must be set to zero.

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	Reserved	Reserved	Reserved	Reserved	Overpressure High Voltage Off	No Ignition Detected	Electronics Warning	Reserved

### 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 Pressure/Vacuum Gauge device profile. The following table provides definitions for each Sensor Alarm bit. Reserved bits must be set to zero.

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	Reserved	Reserved	Reserved	Reserved	Overpressure High Voltage Off	Over Temperature of Electronics	Electronics Failure	Reserved

### Status Extension

8 Bits providing the current sensor alarm state of the object.

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

## SERVICES

The following instance-level services are defined for this subclass of the S-Analog Sensor object:

Service Code	Need in Implementation		Service Name	Description of Service
	Class	Instance		
62 <sub>hex</sub>	n/a	Required	Set High Voltage State	Sets the high voltage to the state specified by the HighVoltageState parameter.

63 <sub>hex</sub>	n/a	Required	Clear Overpressure High Voltage Off Alarm	Clears the Overpressure High Voltage Off bit after an automatic high voltage off
-------------------	-----	----------	---	--

### 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.

### 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.

#### 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

## BEHAVIOR

### Alarms/Warnings

If any bit of the attribute "Sensor Alarm" is set, the high voltage is switched off automatically.

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.

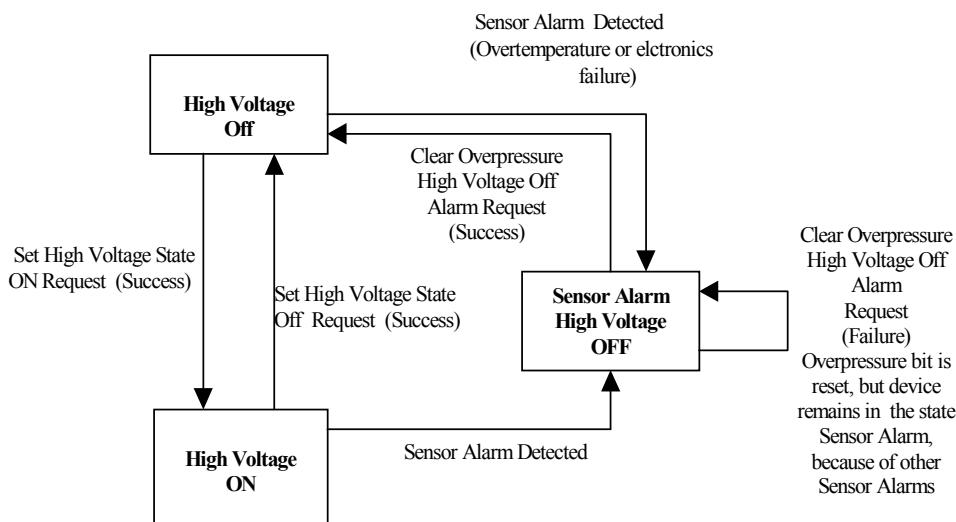


Figure 5-36.1. S-Analog Sensor Object Instance Behavior

See the S-Device Supervisor object (Chapter 5-35) for more information on the Executing State.

**Table 5-36.2. 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: <b>ONLY</b> 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>)

## 5-36.11. S-ANALOG SENSOR OBJECT INSTANCE SUBCLASS 05 (HOT CATHODE ION GAUGE)

### OVERVIEW

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.

### 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.

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

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	STRU CT of BYTE BYTE	Bit definitions of Sensor Warnings	[default=0] see Semantics
95	Optional	Get	V	Sensor Alarm	STRU CT 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

### Semantics:

#### 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.

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

### 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.

### 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.

### 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.

<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>
Sensor Filament 8	Sensor Filament 7	Sensor Filament 6	Sensor Filament 5	Sensor Filament 4	Sensor Filament 3	Sensor Filament 2	Sensor Filament 1

### 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.

## 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).

### 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).

### 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:

The *Active Filament* attribute has Set access.

1. 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.
2. 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.

### *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 Pressure/Vacuum Gauge device profile. The following table provides definitions for each Sensor Warning bit. Reserved bits must be set to zero.

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	Reserved	Reserved	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.

### **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 Pressure/Vacuum Gauge device profile. The following table provides definitions for each Sensor Alarm bit. Reserved bits must be set to zero.

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	Reserved	Reserved	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.

### **Status Extension**

8 Bits providing the current sensor alarm state of the object.

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

## **SERVICES**

The following instance-level services are defined for this subclass of the S-Analog Sensor object:

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	Required	Set Emission State	Turns the filament on and off according to the parameter Emission State
63 <sub>hex</sub>	n/a	Required	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 Emission State ON can be applied.

### 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.

### Set Emission State

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

### Set Degas State

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

## BEHAVIOR

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*.

Sensor Alarm Byte 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*.

**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*.

**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*.

**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.

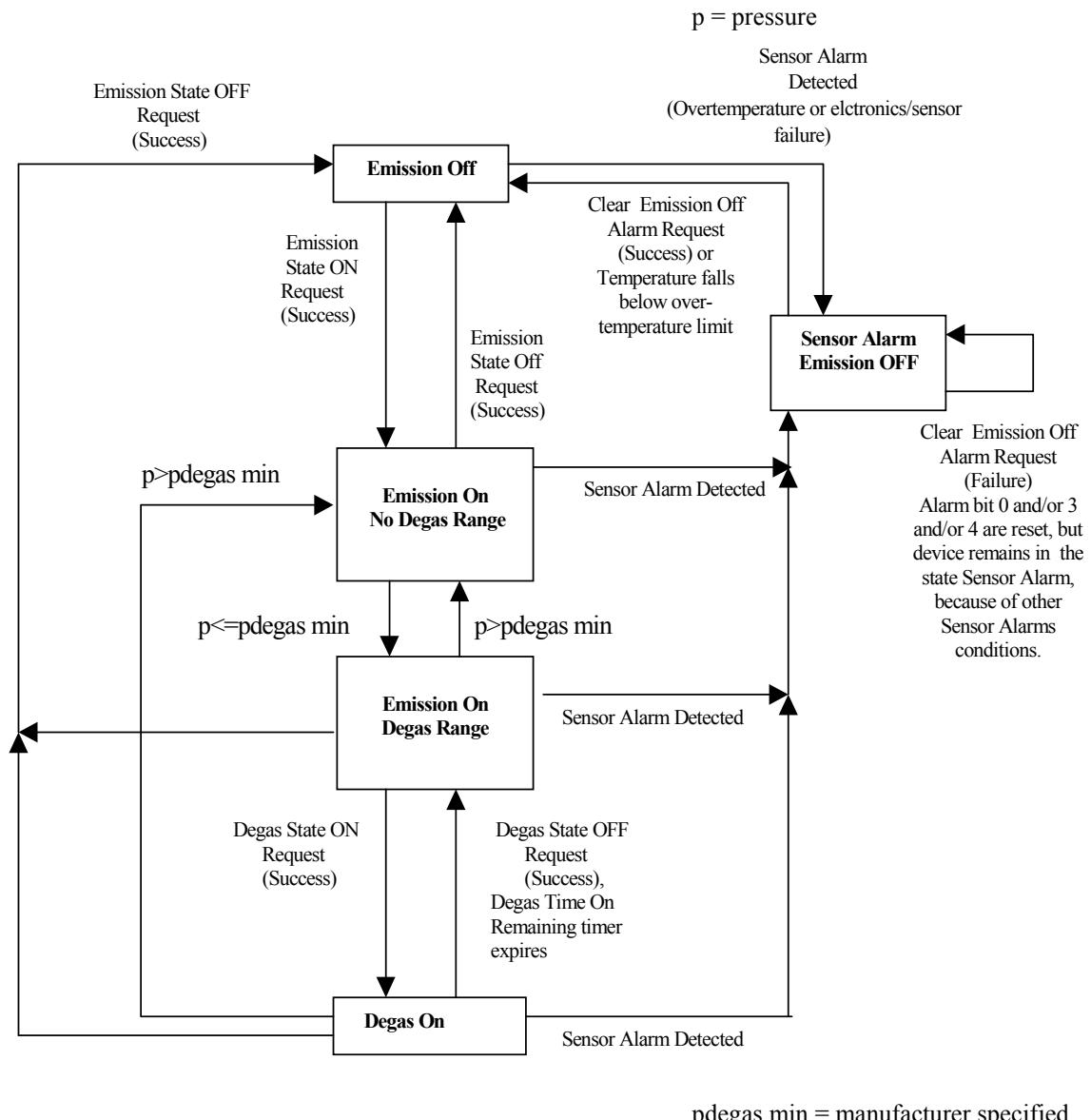
**Bit 4: Environment Failure**

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.

**Sensor Warning Byte 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.

### 5-36.11.1.S-Analog Sensor Object Sub-States for the Executing State



**Figure 5.36.4. S-Analog Sensor Object Instance Behavior**

See the S-Device Supervisor object (Volume 1, Chapter 5-35) for more information on the Executing State.

### 5-36.11.2. S-Analog Sensor Sub-State Event Matrix

**Table 5-36.5. 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

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	Error OSC **	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)	Error AIRS *	Error OSC **	Error OSC **	Not Applicable	Error OSC **
Set Emission State On Request and current pressure $\leq$ minimum degas pressure	Transition to <i>Emission ON Degas Range</i>	Error AIRS *	Error AIRS *	Error OSC **	Error AIRS *
Set Emission State On Request and current pressure $>$ minimum degas pressure	Transition to <i>Emission ON No Degas Range</i>	Error AIRS *	Error AIRS *	Error OSC **	Error AIRS *
Set Emission State On Request at a broken filament in case of user filament selection	Error OSC **	Error AIRS *	Error AIRS *	Error OSC **	Error OSC **
Emission State On Request all filaments broken	Not Applicable	Not Applicable	Not Applicable	Error OSC **	Not Applicable
Current Pressure $\leq$ minimum degas pressure	Ignore Event	Transition to <i>Emission ON Degas Range</i>	Ignore Event	Ignore Event	Ignore Event
Current Pressure $\leq$ 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>

EVENT	SUB – STATE				
	Emission OFF	Emission ON No Degas Range	Emission ON Degas Range	Sensor Alarm Emission Off	Degas On
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>)

## 5-37. 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.

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

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.

### 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.

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

Attr ID	Need in Implementation	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
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 [default] = INT
4	Optional	See Semantics	NV	Data Units	ENGUNITS	Determines the context of <i>Value</i>	see Semantics section [default] = Counts
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]
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 [default] = REAL
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]

Attr ID	Need in Implementation	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
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
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						

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 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.

## 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.

## Semantics:

### Data Type

All Data Type attributes, including *Data Type* and *Gain Data Type*, use the enumerated values specified in Appendix C.

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.

### 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*.

### 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.

### Status

A bit mapped byte which indicates the Alarm and Warning Exception status of the object instance. The following definition applies:

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

### 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.

### 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.

Attribute Value	State
0	Normal
1	Off / Closed
2	On / Open
3	Hold
4	Safe State
5-63	reserved
64-127	Device Specific
128-255	Vendor Specific

### 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:

Attribute Value	State
0	Zero / Off / Closed
1	Full Scale / On / Open
2	Hold Last Value
3	Use Safe Value
4-63	reserved
64-127	Device Specific
128-255	Vendor Specific

### 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:

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.

## 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 5-35.5.

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*.

### **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.

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

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.

### 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.

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

Attr ID	Need in Implementation	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
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 [default] = INT
4	Optional	See Semantics	NV	Data Units	ENGUNITS	Determines the context of the Process related variables such as Setpoint and Process Variable	See Appendix K [default] = Counts
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 [default] = INT
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
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

Attr ID	Need in Implementation	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
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 <i>Setpoint</i> must equal the <i>Process Variable</i>	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 <i>Setpoint</i> must equal the <i>Process Variable</i>	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
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

- \* 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.

## 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.

## Semantics:

### 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.

### 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 K 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.

### 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.

## Status

A bit mapped byte which indicates the Alarm and Warning Exception status of the object instance. The following definition applies:

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

## 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.

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

## 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:

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

### 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.

## 5-38.3. Common Services

The S-Single Stage Controller Object provides the following 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.

## 5-38.4. Object-Specific Services

The S-Single Stage Controller Object provides no Object-Specific services.

## 5-38.5. 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:

### 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).

### 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.

### 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.

### 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.

## 5-39. S-GAS CALIBRATION OBJECT

**Class Code: 34<sub>hex</sub>**

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

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.

### 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.

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	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
6	Optional	Get	NV	Full Scale	STRUCT of: REAL	Full Scale of the device using this object instance Amount	see Semantics section [default] = 0, 0  The amount of measured parameter corresponding to full scale.

Attr ID	Need in Implementation	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
					ENGUNITS	Units	The units for the above. see Data Units Appendix K
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 the value of Subclass is 00, then this attribute is OPTIONAL in implementation, otherwise, this attribute is REQUIRED.

## 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.

## Semantics:

### 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.

### 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.

### 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.

### 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).

### 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)\*, 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”.

\* (Gas Standard Number)

### 5-39.3. Common Services

The S-Gas Calibration Object provides the following Common Services:

Service Code	Need in Implementation		Service Name	Description of Service
	Class	Instance		
0E <sub>hex</sub>	Required	Required	Get_Attribute_Single	Returns the contents of the specified attribute.
10 <sub>hex</sub>	Required	Required	Set_Attribute_Single	Modifies an attribute value.

### 5-39.4. 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

### 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	
		UINT	Valid Sensor Instance	

## 5-39.5. S-Gas Calibration Object Behavior

The behavior of this object is managed by the Device Supervisor Object, defined in Section 5-35.5.

## 5-39.6. 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.

### INSTANCE ATTRIBUTES

The following Instance Attributes are specified for this object subclass 01.

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	The Standard Pressure with respect to the calibration conditions. Default = 101.32
96	Optional	Get	Calibration Temperature	REAL	The Gas Temperature in Degrees C	The Standard Temperature with respect to the calibration conditions. Default = 0.0

### SERVICES

There are no additions or restrictions to the Object Services for this object subclass.

### BEHAVIOR

There are no additions or restrictions to the Behavior for this object subclass.

## 5-40. TRIP POINT OBJECT

**Class Code: 35hex**

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

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.

### 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

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

<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>
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 High 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]

Attr ID	Need in Implementation	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
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
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

## 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.

## Semantics:

### *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.

### ***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 assumes the use of an internal Timer:

For *Status* not set:

- If (*Input*  $\geq$  *Trip Point High*) and (Timer not running)
  - Then start a Timer
  - If (*Input*  $\geq$  *Trip Point High*) and (Time  $\geq$  *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  $\geq$  *Delay*)
    - Then clear *Status*
  - If (*Input*  $\geq$  *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.

### ***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 J for Data Type definitions and Appendix K for Data Units (ENGUNITS) definition.

### ***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.

#### **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*.

#### **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*.

### **5-40.3. Common Services**

The Trip Point Object provides the following 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.

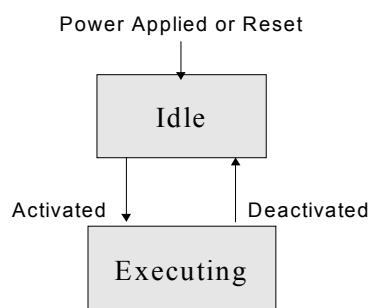
### **5-40.4. Object-Specific Services**

The Trip Point Object provides no Object-Specific services.

### **5-40.5. Behavior**

#### **5-40.5.1 Trip Point Object States**

Figure 5-40.1. Trip Point Object State Transition Diagram



**Table 5-40.2. 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.

### 5-40.5.2 Trip Point Object State Event Matrix

**Table 5-40.3. 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. CONNECTION CONFIGURATION OBJECT

**Class Code: F3hex**

This object defines an interface used to create, configure and control CIP connections on a host device. This specification does not define or constrain the operation of the host device's connection management state machine.

### 5-42.1. Class Attributes

Attribute ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute
1	Required	Get	Revision	UINT	Second revision, value = 2
2	Required	Get	Max Instance	UDINT	Maximum instance number
3	Required	Get	Num Instances	UDINT	Number of connections currently instantiated
4 thru 7	These class attributes are optional and are described in Chapter 4 of this specification.				
8	Required	Get	Format Number	UINT	This number determines the format of instance attribute 9. Format numbers in the range 0-99 are to be defined in this standard. Format_numbers in the range 100-199 are vendor specific. All other format_numbers are reserved and shall not be used.
9	Required	Set	Edit Signature	UDINT	Created and used by configuration software to detect modifications to the instance attribute values

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\_Attribute\_All response and Set\_Attribute\_all request to include attribute 11.

### 5-42.2. Instance Attributes

Attr ID	Need in Implementation	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
1	Required	Get	V	Connection Status	Struct of	When a connection is not OPEN, this attribute tells you why.	See Table 5-262 for details
				gen_status	USINT	General status	
				reserved	USINT	Reserved	Shall be zero
				ext_status	UINT	Extended status	

Attr ID	Need in Implementation	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
2	Required	Set	NV	Connection Flags	WORD	Connection flags	See Table 5-263
3	Required	Set	NV	Target Device ID	Struct of	Device identification used by configuration software to identify target device associated with this instance	
				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 <sup>1</sup>	Set	NV	CS Data Index Number	UDINT	This connection's ControlNet Scheduling read data connection_index number as assigned by the configuration software.	
5	Required	Set	NV	Net Connection Parameters	Struct of		
				conn_timeout	USINT	Connection Timeout Multiplier	
				xport_class_and_t rig	BYTE	Transport Class and Trigger	
				rpi_OT	UDINT	Originator to Target Requested Packet Interval	
				net_OT	UINT	Originator to Target network connection parameters	
				rpi_TO	UDINT	Originator to Target Requested Packet Interval	
				net_TO	UINT	Originator to Target network connection parameters	
6	Required	Set	NV	Connection Path <sup>2</sup>	Struct of		
				open_path_size	USINT	Open connection path size	Number of 16 bit words.
				reserved	USINT	Reserved	Shall be zero
				open connection path	Padded EPATH	Connection path used in the Forward Open service of the Connection Manager.	
7	Required	Get	NV	Config #1 Data <sup>2</sup>	Struct of		

Attr ID	Need in Implementation	Access Rule	NV	Name	Data Type	Description of Attribute	Semantics of Values
				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		
				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	Implementation Defined Attribute	Struct of		
				format_number	UINT	This number determines the format of this attribute.	0-99 Reserved 100 – 199 Vendor Specific All other values are reserved.
				impl_defined_data_size	UINT	Size, in bytes, of implementation defined attribute data that follows	
				impl_defined_data	Array of octet	Implementation defined attribute data specific to a particular implementation.	
10	Required	Set	NV	Config #2 Data <sup>2</sup>	Struct of		
				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		
				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	

<sup>1</sup> This attribute is required for a device on ControlNet; otherwise this attribute shall not be supported.

<sup>2</sup> The data from 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.

**Table 5-42.1. Connection Status Values**

<b>General Status</b>	<b>Description</b>	<b>Extended Status</b>	<b>Description</b>
0x00	If originator-side connection entry, then Success	Various	Connection is open
	Else if target-side connection entry, then see Extended Status for more information	0x00	Connection is idle
		0x01 or greater	If producing data, then number of peer consumers
0x01	Connection Failed	Various	Extended status as defined by part 4, Network and Transport Layer
Various (0x02 – x26)	Other general status values	Various	Extended status as defined by part 4, Network and Transport Layer
0xD0	Connection configuration object-specific general status	0x0001	Connection is closed or stopped (via the Close or Stop services)
		0x0002	Connection open is pending
		0x0003	Connection close is pending

**Table 5-42.2. Connection Flags Attribute Definition**

<b>Bit</b>	<b>Meaning</b>
0	Connection 0 = Originator 1 = Target
1 – 3	O⇒T Real time transfer format 0 = Use 32-bit Run/Program header to indicate idle mode. 1 = Use zero data length packet to indicate idle mode. 2 = None. Connection is pure data and is modeless. 3 = Heartbeat. Other values are reserved for future use.
4 - 6	T⇒O Real time transfer format 0 = Use 32-bit Run/Program header to indicate idle mode. 1 = Use zero data length packet to indicate idle mode. 2 = None. Connection is pure data and is modeless. 3 = Heartbeat. Other values are reserved for future use.
7 - 15	Reserved

### 5-42.3. Common Services

The Connection Configuration Object provides the following Common Services:

Service Code	Need in Implementation		Service Name	Description of Service
	Class	Instance		
01_hex	Required	Required	Get_Attribute_All	Gets all attributes of the specified instance.
02_hex	Optional	Required	Set_Attribute_All	Sets all attributes of the specified instance.
08_hex	Required	N/A	Create	Creates a new connection instance.
09_hex	Required	Required	Delete	Deletes an existing connection instance.
0E_hex	Optional	Required	Get_Attribute_Single	Returns the contents of the specified attribute.
10_hex	Required	Optional	Set_Attribute_Single	Modifies an attribute value.
15_hex	Required	Required	Restore	Restore current connection attributes.

### 5-42.4. Object Specific Services

The Connection Configuration Object provides the following 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	Optional	Optional	Open	Opens connections
4Dhex	Optional	Optional	Close	Closes connections
4Ehex	Optional	Optional	Stop	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

### 5-42.2.1 Get\_Attribute\_All (Service Code 0x01)

This service shall read all attributes associated with an existing instance and shall support the following error codes:

General Status	Extended Status	Error Description
0x04	None	path syntax error
0x05	None	Instance undefined
0x08	None	Unimplemented service

The structure of the Get\_Attribute\_All response is shown in the table below.

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

Name	Data Type	Description
	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 CM_open_request 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.
Implementation Defined Attribute	Struct of	The implementation defined attribute.
	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

#### 5-42.4.2 Set\_Attribute\_All (Service Code 0x02)

This service shall write all attributes associated with an existing instance and shall return the following error codes:

General Status	Extended Status	Error Description
0x04	None	path syntax error
0x05	None	Instance undefined
0x08	None	Unimplemented service
0x0C	None	Wrong object state

The structure of the Set\_Attribute\_All response is shown in the table below.

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

Name	Data Type	Description
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 Name	Struct of	The connection name.
	USINT	Number of characters in connection name.
	USINT	Pad.
	STRING2	Connection name encoded in UNICODE.
Implementation Defined Attribute	Struct of	The implementation defined attribute.
	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

### 5-42.4.3 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:

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
	UINT	Attribute value

This service shall support the following error codes:

General Status	Extended Status	Error Description
0x02	0x0001	Insufficient resource — maximum number of instances already exist
	0x0002	Insufficient resource — not enough memory on device
0x03	None	Invalid parameter error (when attribute count is invalid)
0x04	None	path syntax error
0x0C	None	Wrong object state
0x0E	None	Attempt to set a read-only attribute
0x13	None	Insufficient request data — request was too short or truncated
0x1C	None	Attribute list shortage — required attribute was missing

### 5-42.4.4 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 shall support the following error codes:

General Status	Extended Status	Error Description
0x04	None	path syntax error
0x05	None	Instance undefined
0x08	None	Unimplemented service
0x0C	None	Wrong object state

### 5-42.4.5 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. If addressed to a specific instance, only modifications for that instance shall be discarded.

This service shall support the following error codes:

General Status	Extended Status	Error Description
0x04	None	path syntax error
0x05	None	Instance undefined
0x0C	None	Wrong object state

#### 5-42.4.6 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 effecting the state of the Connection Configuration object. If the edit watchdog timer expires, all pending modifications shall be discarded.

This service shall support the following error codes:

General Status	Extended Status	Error Description
0x04	None	path syntax error
0x05	None	Instance undefined

#### 5-42.4.7 Open (Service Code 0x4C)

The Open service shall cause the connection associated with an instance of the Connection Configuration object to open. If the Open 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.

This service shall support the following error codes:

General Status	Extended Status	Error Description
0x04	None	path syntax error
0x05	None	Instance undefined
0x08	None	Unimplemented service

#### 5-42.4.8 Close (Service Code 0x4D)

The Close service shall cause the connection associated with an instance of the Connection Configuration object to close. If the Close 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 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 service is issued.

This service shall support the following error codes:

General Status	Extended Status	Error Description
0x04	None	path syntax error

General Status	Extended Status	Error Description
0x05	None	Instance undefined
0x08	None	Unimplemented service

#### 5-42.4.9 Stop (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 an 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 service request is issued.

This service shall support the following error codes:

General Status	Extended Status	Error Description
0x04	None	path syntax error
0x05	None	Instance undefined
0x08	None	Unimplemented service

#### 5-42.4.10 Change\_Start (Service Code 0x4F)

This service shall:

- signal the beginning of an edit session;
- synchronise 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).

This service shall support the following error codes:

General Status	Extended Status	Error Description
0x04	None	path syntax error
0x0C	None	Wrong object state
0x10	None	Device state conflict

#### 5-42.4.11 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.

The following request parameter is defined:

Parameter	Data Type	Description
Starting Instance	UDINT	Starting instance number

The following response parameters are defined for this service:

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 shall support the following error codes:

General Status	Extended Status	Error Description
0x03	None	Invalid parameter error (when attribute count is invalid)
0x04	None	path syntax error
0x08	None	Unimplemented service

#### 5-42.4.12 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.

The following request parameter is defined:

Parameter	Data Type	Description
Change Type	UINT	0 = Full 1 = Incremental All other values reserved.

This service shall support the following error codes:

General Status	Extended Status	Error Description
0x02	None	Resource unavailable. Indicates that there is not enough memory on the host device to support the specified configuration.
0x04	None	path syntax error
0x0C	None	Wrong object state
0x10	None	Device state conflict

### 5-42.4.13 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:

Parameter	Data Type	Description
Change Type	UINT	0 = Full 1 = Incremental All other values reserved.

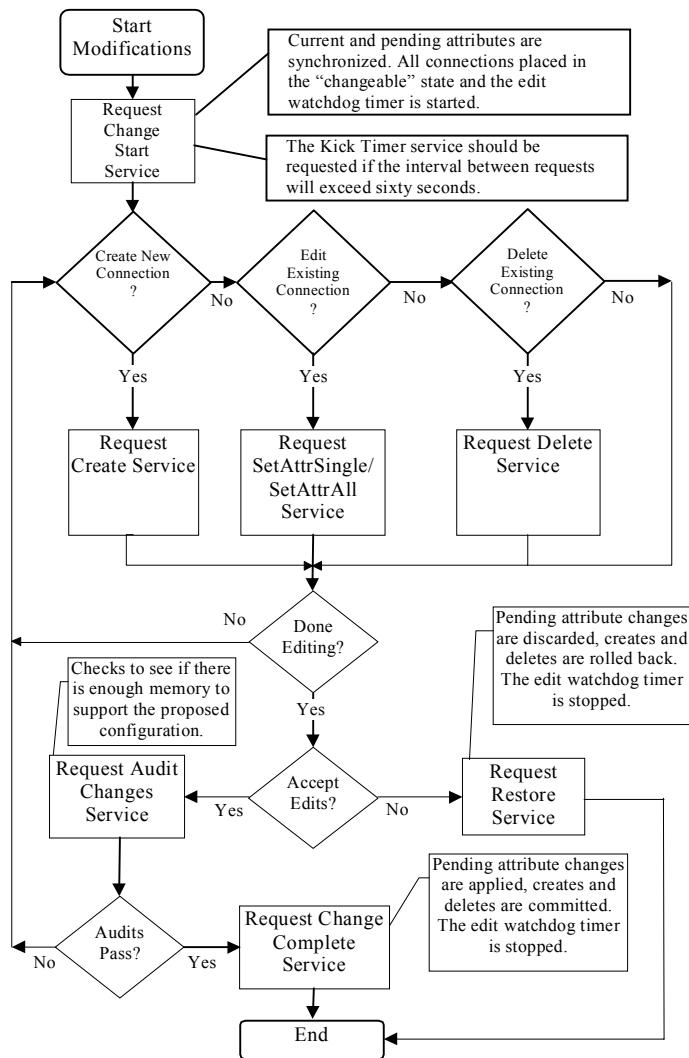
This service shall support the following error codes:

General Status	Extended Status	Error Description
0x02	None	Resource unavailable. Indicates that there is not enough memory on the host device to support the specified configuration.
0x04	None	path syntax error
0x0C	None	Wrong object state
0x10	None	Device state conflict

## 5-42.5. Behavior

The flowchart below summarizes the process of creating, editing, and deleting connections.

**Figure 5-42.3. Connection Configuration Object Edit Flowchart**



This page is intentionally left blank

## **Volume 1: CIP Common Specification**

### **Chapter 6: Device Profiles**

---

This page is intentionally left blank

## 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.

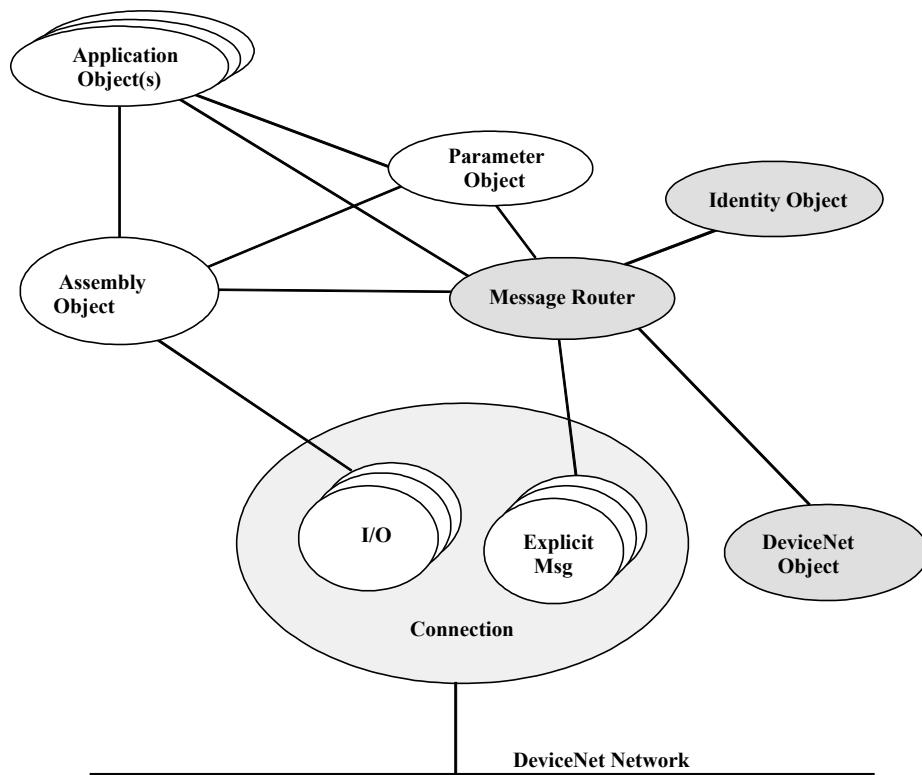
## 6-2. THE OBJECT MODEL

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:

- Connection Object Class or Connection Manager Object Class
- Network specific link object (eg. DeviceNet, ControlNet, TCP/IP Object Class)
- Identity Object Class
- Message Router Object Class

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.

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 6-2.2. 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.

As you can see in the Flow Transmitter example in Figure 6.2., the objects in this device have the following interfaces:

**Table 6-2.3. Flow Transmitter Example: 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 the example Flow Transmitter device.

**Table 6-3.1. Flow Transmitter Example: Identifying I/O Assembly Instances**

Number	Type	Name
1	Input	Basic Input

### 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. See Table 2.D \$\$.

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. Flow Transmitter 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.

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 6-4.1. Parameter Instance Listing**

Instance Number	Configuration Parameter Name
X	Parameter name

**Table 6-4.2. Configuration Parameter Mapping**

Configuration Parameter Name	Class		Instance Number	Attribute		Data Type
	Name	Number		Name	Number	
Parameter name	Class	xx <sub>hex</sub>	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 6-4.3. 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 6-4.4. 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:

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.

For information about:	Go to section:	Device Type Number:
AC Drives	6-15	02 <sub>hex</sub>
Barcode Scanner	6-17	Not yet assigned
Circuit Breaker	6-25	Not yet assigned
Communications Adapter	6-13	0C <sub>hex</sub>
Contactor	6-27	15 <sub>hex</sub>
Control Station	6-23	Not yet assigned
ControlNet Physical Layer	6-34	32 <sub>hex</sub>
ControlNet Programmable Logic Controller	6-33	0E <sub>hex</sub>
DC Drives	6-15	13 <sub>hex</sub>
Encoder	6-21	Not yet assigned
General Purpose Analog I/O	6-14	Not yet assigned
General Purpose Discrete I/O	6-12	07 <sub>hex</sub>
Generic Device	6-8	00 <sub>hex</sub>
Human-Machine Interface	6-30	18 <sub>hex</sub>
Inductive Proximity Switch	6-10	05 <sub>hex</sub>
Limit Switch	6-9	04 <sub>hex</sub>

<b>For information about:</b>	<b>Go to section:</b>	<b>Device Type Number:</b>
Mass Flow Controller	6-31	1A <sub>hex</sub>
Message Display	6-24	Not yet assigned
Motor Overload	6-19	03 <sub>hex</sub>
Motor Starter	6-28	16 <sub>hex</sub>
Photoelectric Sensor	6-11	06 <sub>hex</sub>
Pneumatic Valve(s)	6-26	1B <sub>hex</sub>
Position Controller	6-18	10 <sub>hex</sub>
Resolver	6-22	09 <sub>hex</sub>
Servo Drives	6-16	Not yet assigned
Soft Start	6-29	17 <sub>hex</sub>
Weigh Scale	6-20	Not yet assigned
Vacuum Pressure Gauge	6-32	1C <sub>hex</sub>

The following device type numbers have been obsoleted.

<b><u>Obsolete Device Type Number:</u></b>	<b><u>Previous Profile Assignment:</u></b>
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: 00hex

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.

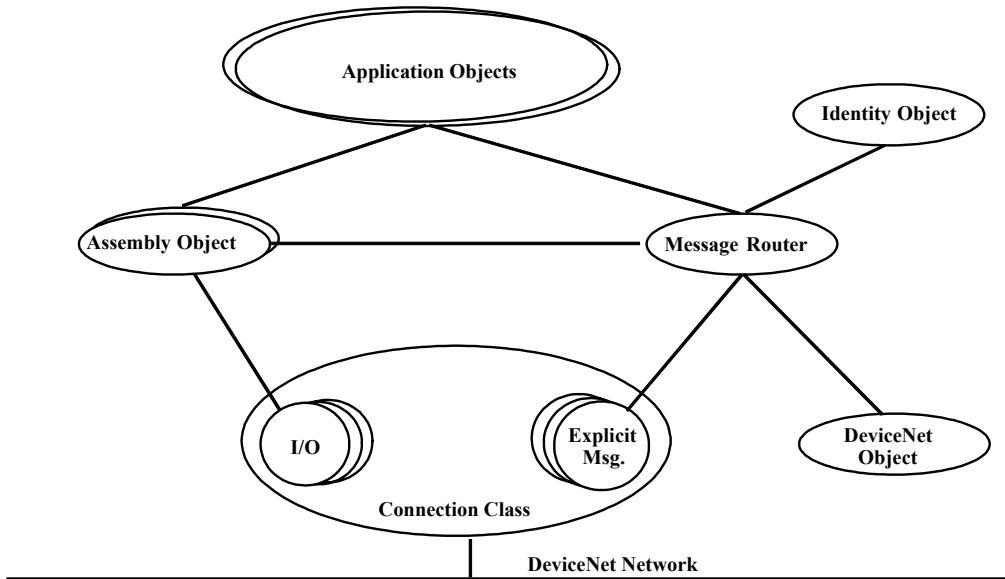
**6-8.1.****Object Model**

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

Object Class	Optional/Required	# of Instances
Identity	Required	at least 1
Message Router	Required	1
Network Specific Link Object	Required	at least 1
Connection	Required	at least 1 I/O and 1 explicit
Assembly	Required	at least 1
Application	Required	at least 1

The Generic 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 Chapter 2, Contents of a Device Profile.

**Figure 6-8.1. 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.

Object	Effect on behavior
Identity	Supports the Reset service
Message Router	No effect
Network Specific Link Object	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
Application	Defines device operation

## 6-8.3. Defining Object Interfaces

The objects in the Generic Device have the interfaces listed in the following table:

Object	Interface
Identity	Message Router
Message Router	Explicit Messaging Connection Instance
Network Specific Link Object	Message Router
Connection Class	Message Router
Assembly	I/O Connection or Message Router
Application	Assembly or Message Router

## 6-9. LIMIT SWITCH

Device Type: 04hex

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.

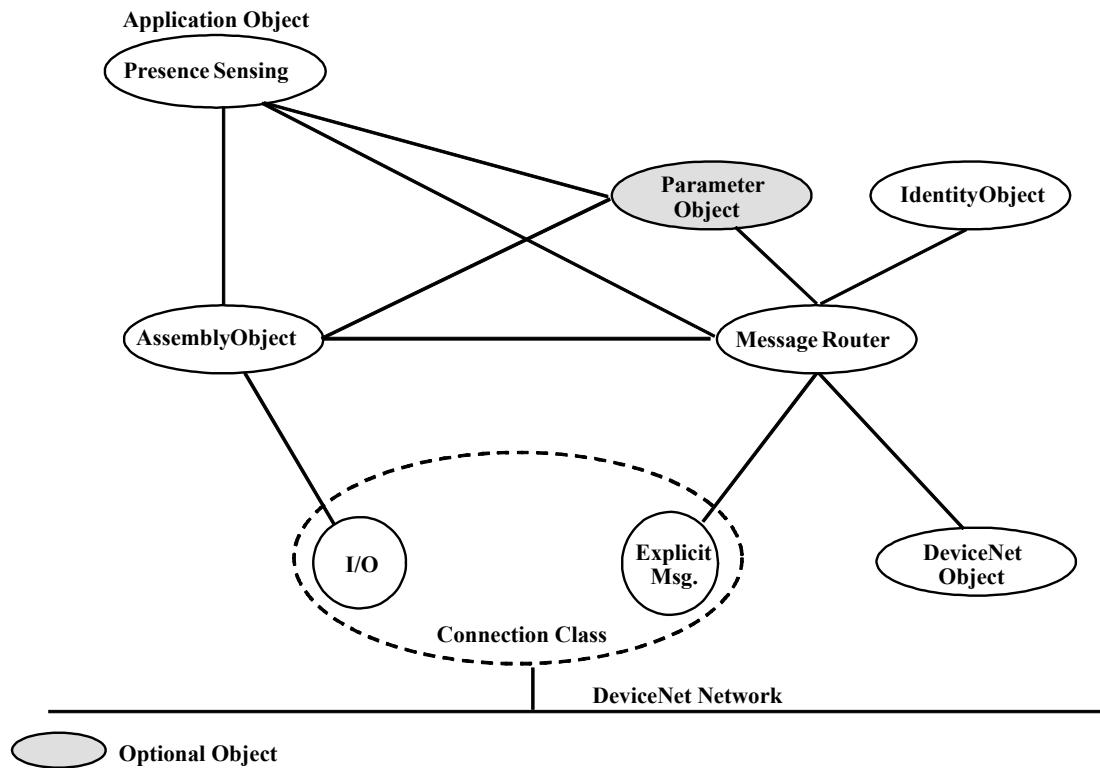
### 6-9.1. Object Model

The Object Model in Figure 6-9.1. 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.

Object Class	Optional/Required	# of Instances
Identity	Required	1
Message Router	Required	1
Network Specific Link Object	Required	1
Connection	Required	2 (explicit,-I/O)
Assembly	Required	1
Parameter	Optional	1
Presence Sensing	Required	1

**Figure 6-9.1.** 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.

Object	Effect on behavior
Identity	Supports the Reset service
Message Router	No effect
Network Specific Link Object	Configures port attributes
Connection	Contains the number of logical ports into or out of the device
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:

Object	Interface
Identity	Message Router
Message Router	Explicit Messaging Connection Instance
Network Specific Link Object	Message Router
Connection	Message Router
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.

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.

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.

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.

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.

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 configure parameters for a limit switch.

```
[Parms]
Param1=
 0,                                $Operate Mode
 2,"20 0e 24 01 30 08",            $Data Placeholder
 0x0002,                            $Path size and Path to Operate Mode Attr
 4,1                                $Descriptor (support enumerated strings
 "Operate Mode",                    $Data Type and Size (Boolean)
 "",                                 $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=
 "Normally Open",                  $Operate Mode Enumerated Strings
 "Normally Closed";               $For value=0
                                     $For value=1
```

## 6-9.8. Effect of Configuration Parameters on Behavior

The configuration parameter affects the device's behavior as shown below.

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 Type: 05hex

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.

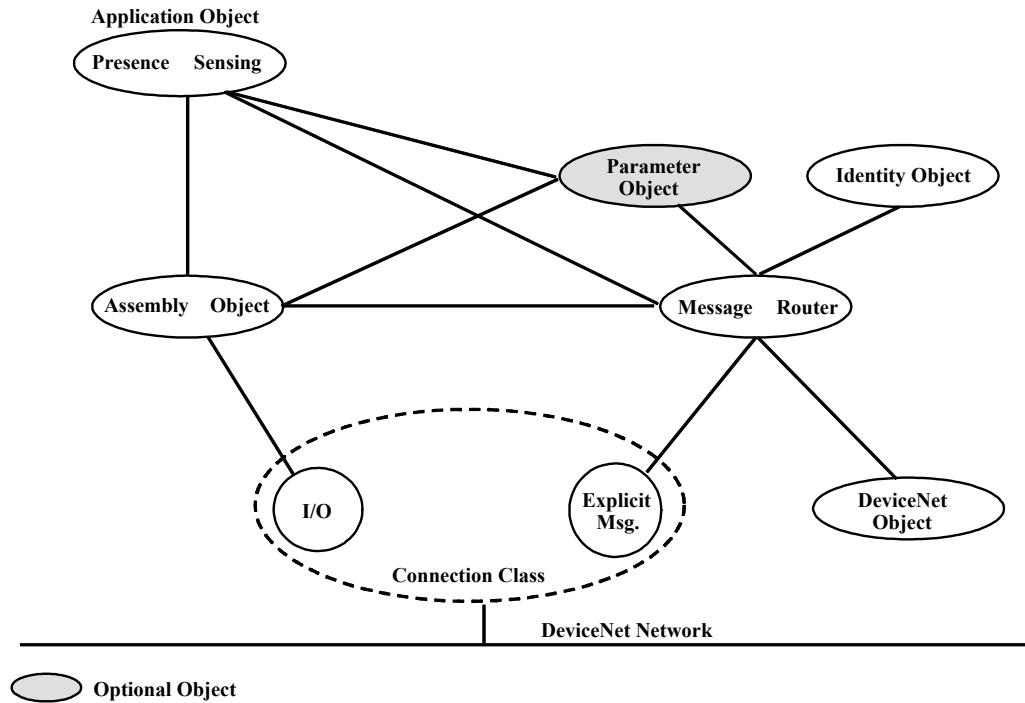
### 6-10.1. Object Model

The Object Model in Figure 6-10.1. 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.

Object Class	Optional/Required	# of Instances
Identity	Required	1
Message Router	Required	1
Network Specific Link Object	Required	1
Connection	Required	2 (explicit, I/O)
Assembly	Required	1
Parameter	Optional	1
Presence Sensing	Required	1

**Figure 6-10.1. 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.

Object	Effect on behavior
Identity	Supports the Reset service
Message Router	No effect
Network Specific Link Object	Configures port attributes
Connection	Contains the number of logical ports into or out of the device
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:

Object	Interface
Identity	Message Router
Message Router	Explicit Messaging Connection Instance
Network Specific Link Object	Message Router
Connection	Message Router
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.

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.

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	0	Reserved	Reserve d	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.

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.

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.

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=
 0,                               $Operate Mode
 3,"20 0e 24 01 30 08",          $Data Placeholder
 0x0002,                          $Path size and Path to Operate Mode Attr
 4,1                             $Descriptor (support enumerated strings)
 "Operate Mode",                 $Data Type and Size (Boolean)
 "",                             $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=
  "Normally Open",               $Operate Mode Enumerated Strings
  "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.

Parameter	Effect on Behavior
Operate Mode	Inverts the level defined for the Output attribute of the Presence Sensing Object

## 6-11. PHOTOELECTRIC SENSOR

Device Type: 06hex

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

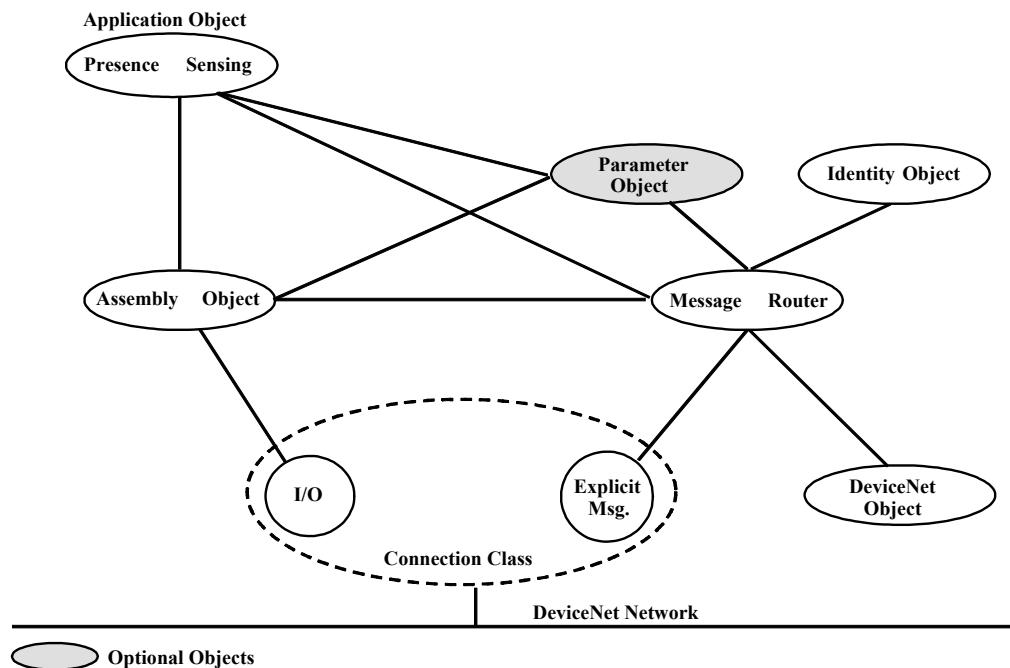
The Object Model in Figure 6-11.1. represents a photoelectric sensor. The table below indicates:

- 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.

Object Class	Optional/Required	# of Instances
Identity	Required	1
Message Router	Required	1
Network Specific Link Object	Required	1
Connection	Required	2 (explicit, I/O)
Assembly	Required	1
Parameter	Optional	1
Presence Sensing	Required	1

**Figure 6-11.1. 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.

Object	Effect on behavior
Identity	Supports the Reset service
Message Router	No effect
Network Specific Link Object	Configures port attributes
Connection	Contains the number of logical ports into or out of the device
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:

Object	Interface
Identity	Message Router
Message Router	Explicit Messaging Connection Instance
Network Specific Link Object	Message Router
Connection	Message Router
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.

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.

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	0	Reserved	Reserve d	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.

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, 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 photoelectric sensor.

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.

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)
4,1	\$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 effects the device's behavior as shown below.

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 Type: 07hex

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 6-12.1. 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

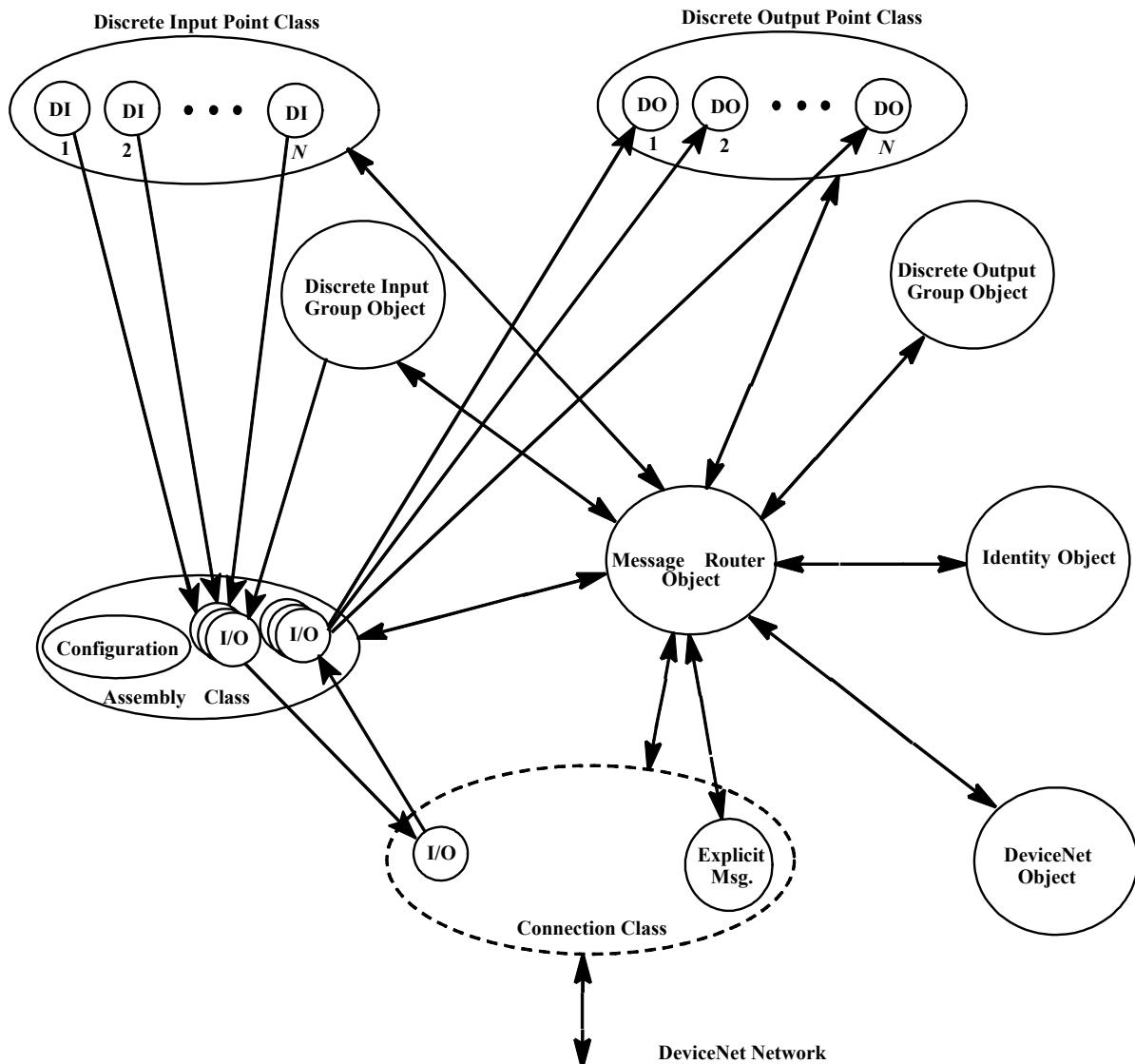
The CIP Object Library provides more details about these objects.

Object Class	Optional/Required	# of Instances
Identity	Required	1
Message Router	Required	1
Network Specific Link Object	Required	1
Connection	Required	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

**Figure 6-12.1. 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.

Object	Effect on behavior
Identity	Supports the Reset service
Message Router	No effect
Network Specific Link Object	Configures port attributes
Connection	Contains the number of logical ports into or out of the device
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:

Object	Interface
Identity	Message Router
Message Router	Explicit Messaging Connection Instance
Network Specific Link Object	Message Router
Connection	Message Router
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

The following table identifies the I/O assembly instances supported by this device.

Number	Type	Name
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
52	Input	2-Point Input with Single Input Status and Single Output Status Bits

Number	Type	Name
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.

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
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
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	Discrete Input N-2	

The I/O Assembly data attribute for the input data with one status bit has the format shown below.

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						
15	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

<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>
	2	Status	Reserved						
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.

<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>
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
26	0	Discrete Input8	Discrete Input7	Discrete Input6	Discrete Input5	Discrete Input4	Discrete Input3	Discrete Input2	Discrete Input1

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
•	•	•	•	•	•	•	•	•	•
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.

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
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
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

<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>
	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.

<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>
41	0	Reserved							Discrete Output Status1
42	0	Reserved							Discrete Output Status1
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
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

<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>
47	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
	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	Discrete Output Status N-2

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.

<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>
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					
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

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	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.

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 Input4	Discrete Input3	Discrete Input2	Discrete Input1
	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
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

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	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.

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 Input2	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	
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						
	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
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

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	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.

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

**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.

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.

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.

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.

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
40	0	Reserved	Reserve d	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.

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.

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 can not 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 can not be configured via this assembly. The default is 0.

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=
0,
6,"20 1E 24 01 30 07",
0x00,
4,
1,
"Fault Action",
"",
"",
0,1,0;
1,1,1,0,0,0,0,0,0;

Param2=
0,
6,"20 1E 24 01 30 09",
0x00,
4,
1,
"Idle Action",
"",
"",
0,1,0;
1,1,1,0,0,0,0,0,0;

[Groups]
```

\$ Fault Action  
\$ reserved  
\$ Link Path Size and Link Path  
\$ No support for settable path,  
\$ enumerated strings, scaling,  
\$ scaling link, or real time  
\$ update of value. Value is  
\$ gettable and settable.  
\$ Data Type  
\$ Data Size  
\$ Parameter Name  
\$ Units String not used  
\$ Help string not used  
\$ Min, Max, and Default values  
\$ Not used

\$ Idle Action  
\$ reserved  
\$ Link Path Size and Link Path  
\$ No support for settable path,  
\$ enumerated strings, scaling,  
\$ scaling link, or real time  
\$ update of value. Value is  
\$ gettable and settable.  
\$ Data Type  
\$ Data Size  
\$ Parameter Name  
\$ Units String not used  
\$ Help string not used  
\$ Min, Max, and Default values  
\$ Not used

\$ No need to support

## 6-13. Communications Adapter

Device Type: 0Chex

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

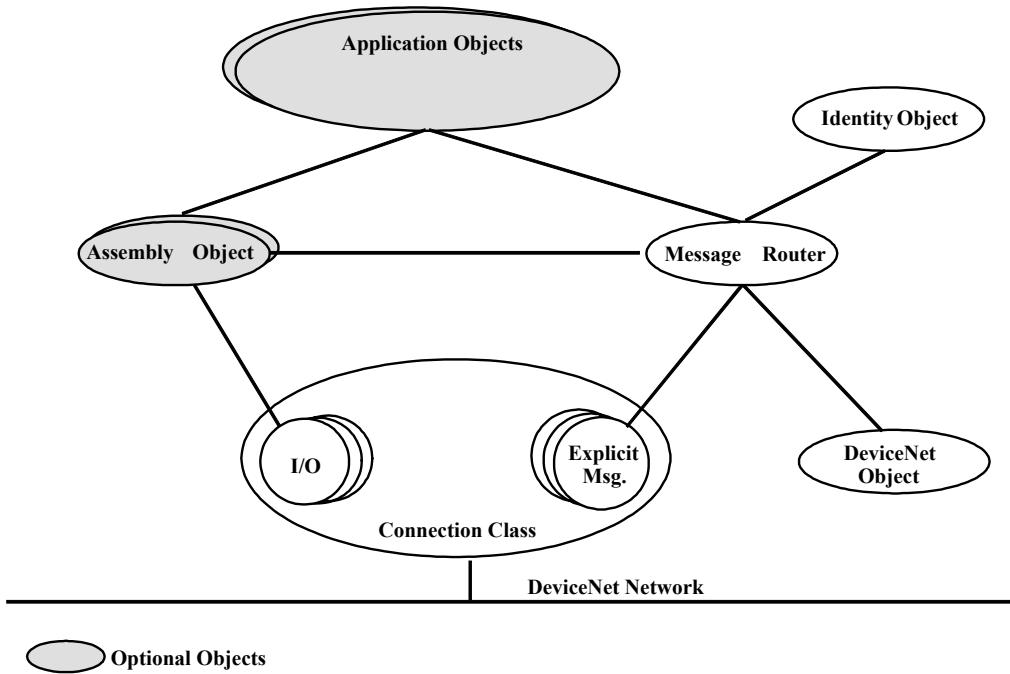
The Object Model in Figure 6-13.1. represents the minimum support in a Communications Adapter. The table below indicates:

- the object classes present in this device
- whether or not the class is required
- the number of instances present in each class

Object Class	Optional/Required	# of Instances
Identity	Required	at least 1
Message Router	Required	1
Network Specific Link Object	Required	at least 1
Connection	Required	at least 1 I/O and 1 explicit
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 Chapter 2 \$\$, Contents of a Device Profile. 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.

**Figure 6-13.1. Object Model for the Communications Adapter**



## 6-14. GENERAL PURPOSE ANALOG I/O

The profile for this device type will be defined by the Open DeviceNet Vendor Association, Inc. and ControlNet International.

## 6-15. AC DRIVES

**Device Type:** 02<sub>hex</sub>

## DC DRIVES

**Device Type:** 13<sub>hex</sub>

These device profiles describe standard objects and behaviour 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 makes 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.

### 6-15.0.1 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.

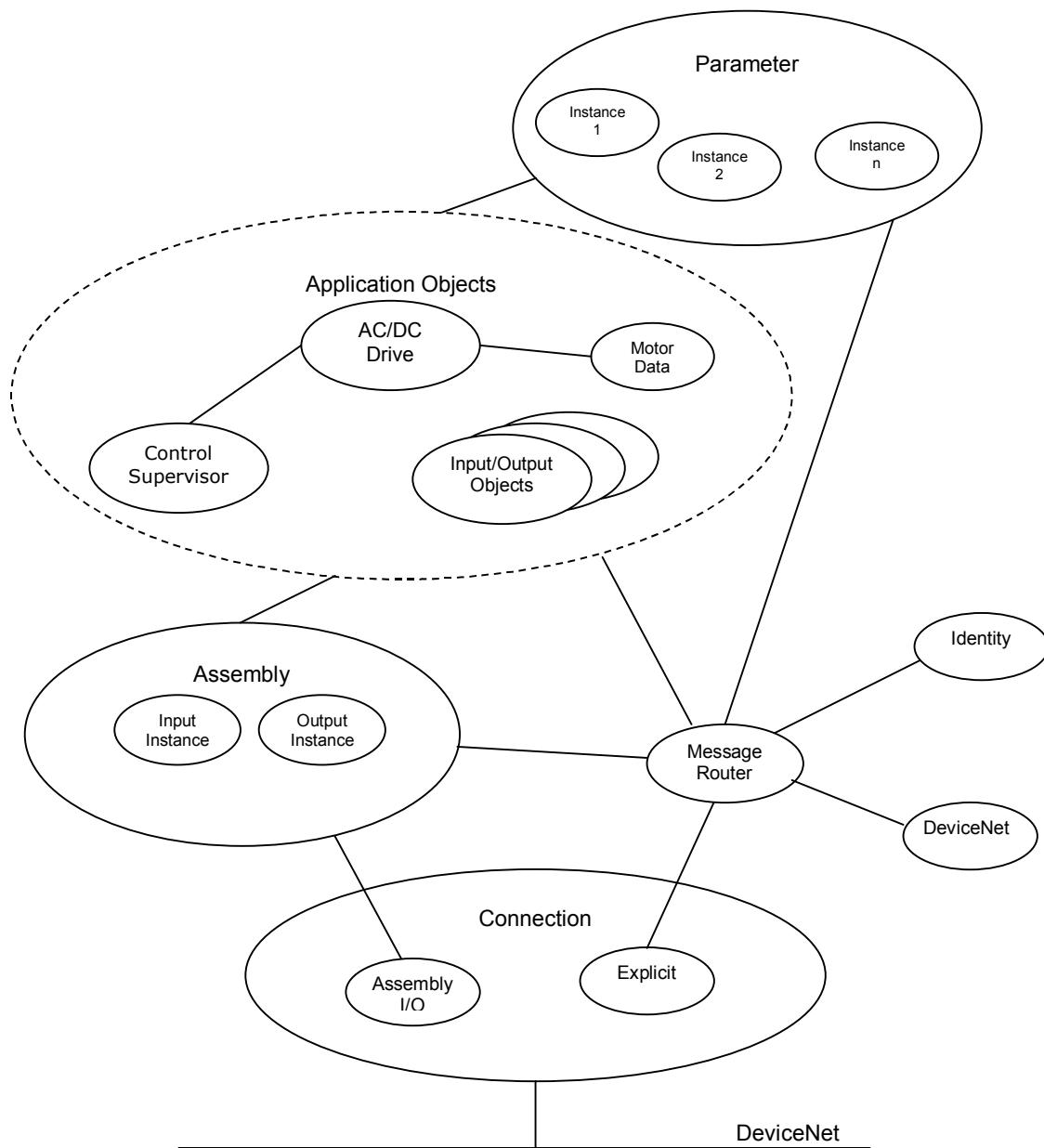
### 6-15.1 Object Model

The Object Model in figure 6-15.1 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.

Object Class	Optional / Required	# of Instances
Identity	Required	1
Message Router	Optional	-
Network Specific Link Object	Required	1
Connection	Required	2
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 6-15.7. Object Model for AC and DC Drives**

## 6-15.2. How Objects Affect Behavior

The objects in this device affect the device's behavior as shown in the following table.

Object	Effect on Behavior
Identity	Supports the Reset service
Message Router	No effect
Network Specific Link Object	Configures port attributes
Connection	Logical ports into or out of the device
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

## 6-15.3. Defining Object Interfaces

The objects in this device have the interfaces listed in the following table.

Object	Interface
Identity	Message Router
Message Router	Explicit Message Connection
Network Specific Link Object	Message Router
Connection	Message Router
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

#### 6-15.4. 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 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.

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
	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.

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

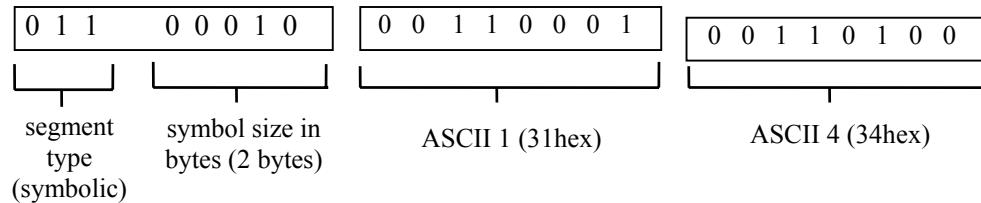
If a bit is not used in an IO Assembly, it is reserved for use in other Assemblies. Reserved bits in Output Assemblies are ignored by the consuming device. Reserved bits in Input Assemblies are set to zero by the producing device.

Reserved bits in the IO Assembly Data Attribute Format Tables are shaded.

### 6-15.4.1. 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.

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).

### 6-15.5. I/O Assembly Data Attribute Format

The I/O Assembly Data Attributes have the format shown below.

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
20	0						Fault Reset		Run Fwd
	1								
	2								Speed Reference (Low Byte)
	3								Speed Reference (High Byte)
21	0		NetRef	NetCtrl			Fault Reset	Run Rev	Run Fwd
	1								
	2								Speed Reference (Low Byte)
	3								Speed Reference (High Byte)
22	0						Fault Reset		Run Fwd
	1								
	2								Speed Reference (Low Byte)
	3								Speed Reference (High Byte)
	4								Torque Reference (Low Byte)
	5								Torque Reference (High Byte)

<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>
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						Running 1		Faulted
	1								
	2								Speed Actual (Low Byte)
	3								Speed Actual (High Byte)
71	0	At Referen ce	Ref From Net	Ctrl From Net	Ready	Runnin g2 (Rev)	Runnin g1 (Fwd)	Warnin g	Faulted
	1								Drive State
	2								Speed Actual (Low Byte)
	3								Speed Actual (High Byte)

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
72	0						Running 1		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	Running 2 (Rev)	Running 1 (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)							
74	0						Running 1		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	Running 2 (Rev)	Running 1 (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)							

## 6-15.6. 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.

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	2A <sub>hex</sub>	1	NetProc	5

Data Component Name	Class		Instance Number	Attribute	
	Name	Number		Name	Number
Drive					
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.

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

## 6-15.7. 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.

### 6-15.7.1. Mapping Parameter Object Data

The following table indicates the Parameter Object data mapping for an AC or DC Drive device.

Configuration Parameter	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

Configuration Parameter	Class		Instance Number	Attribute	
	Name	Number		Name	Number
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
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

### 6-15.7.2. 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.

<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

<b>Parameter Group Name</b>	<b>Instance Number</b>	<b>Parameters in Group</b>
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 DRIVES

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**

The profile for this device type will be defined by the Open DeviceNet Vendor Association, Inc. and ControlNet International

## 6-18. POSITION CONTROLLER

**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 6-12.1 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, The CIP Object Library, provides more details about these objects.

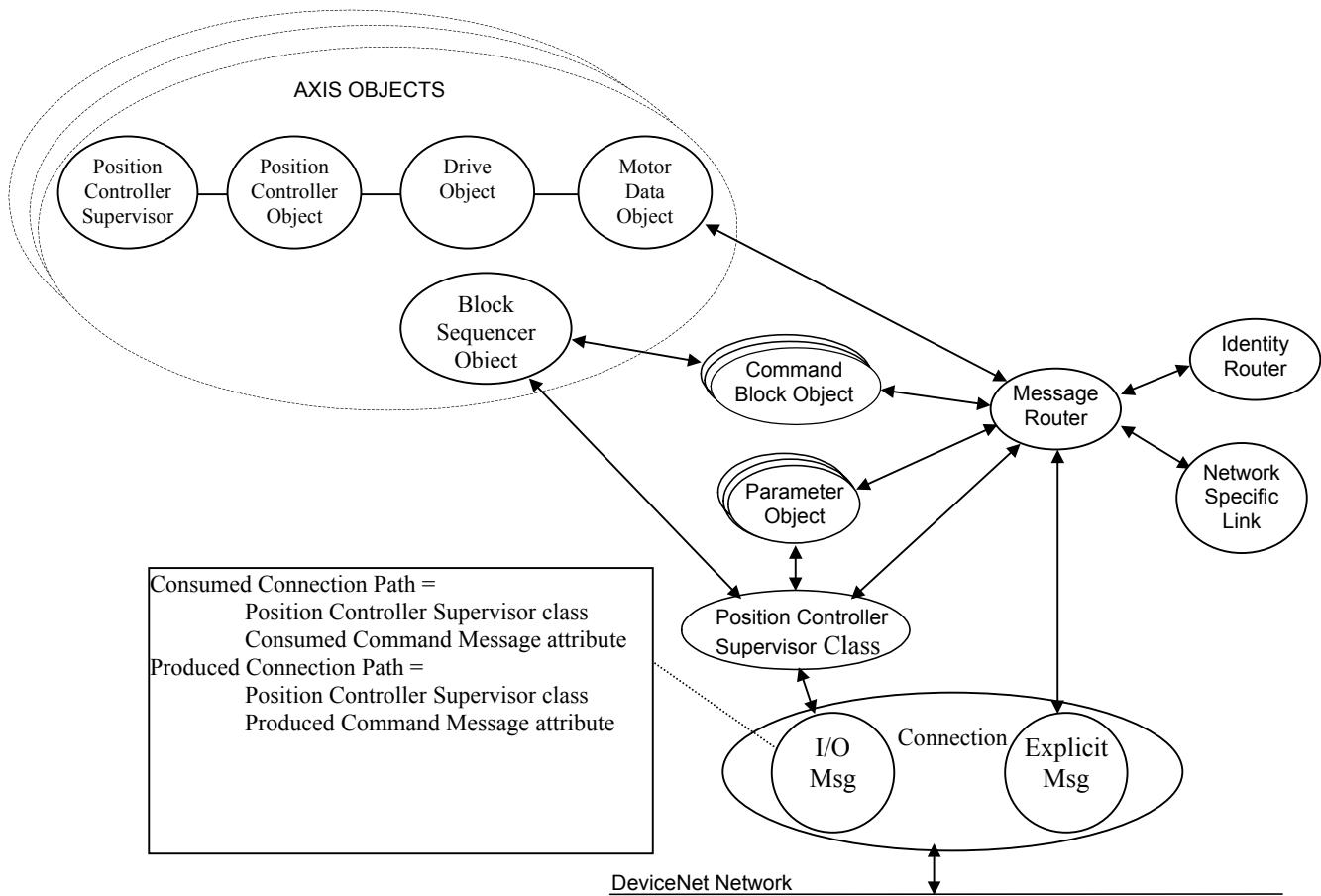
Object Class	Optional / Required	# of Instances
Identity	Required	1
Message Router	Required	1
Network Specific Link Object	Required	1
Connection	Required	2 (explicit/ I/O)
Position Controller Supervisor	Required	1 per Axis
Position Controller	Required	1 per Axis
Command Block	Optional	-
Block Sequencer	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.

**Figure 6-18.1 Object Model for a Position Controller**



## 6-18.2. How Objects Affect Behavior

The object for this device affect the device's behavior as shown in the table below.

Object	Effect on behavior
Identity	Supports the Reset service
Message Router	No effect
Network Specific Link Object	Configures port attributes
Connection	Contains the number of logical ports into or out of the device
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:

Object	Interface
Identity	Message Router
Message Router	Explicit Messaging Connection
Network Specific Link Object	Message Router
Connection	Message Router
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 6-12.1.

#### 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.

**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											

### Response Message Format

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	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

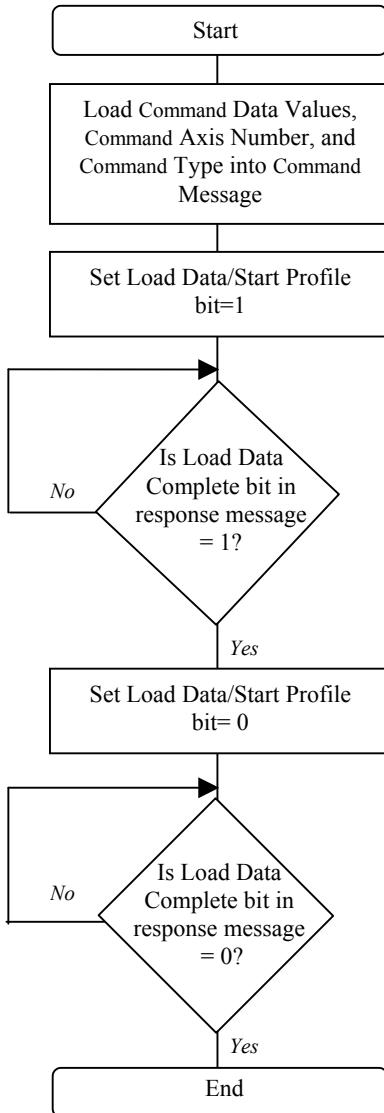
A profile move starts when the command message type for the specified mode is loaded and the Load Data/Start Profile bit transitions from zero to one. The table below shows the command message type which starts a profile move for each mode.

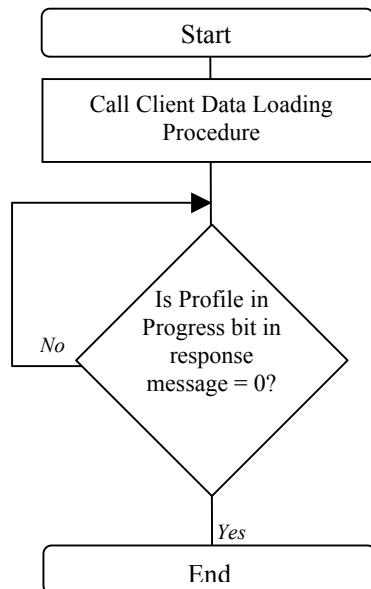
Mode (Attribute 3)	Command Message Type which Starts Motion
0 = Position	01 = Position
1 = Velocity	02 = Velocity
2 = Torque	05 = Torque

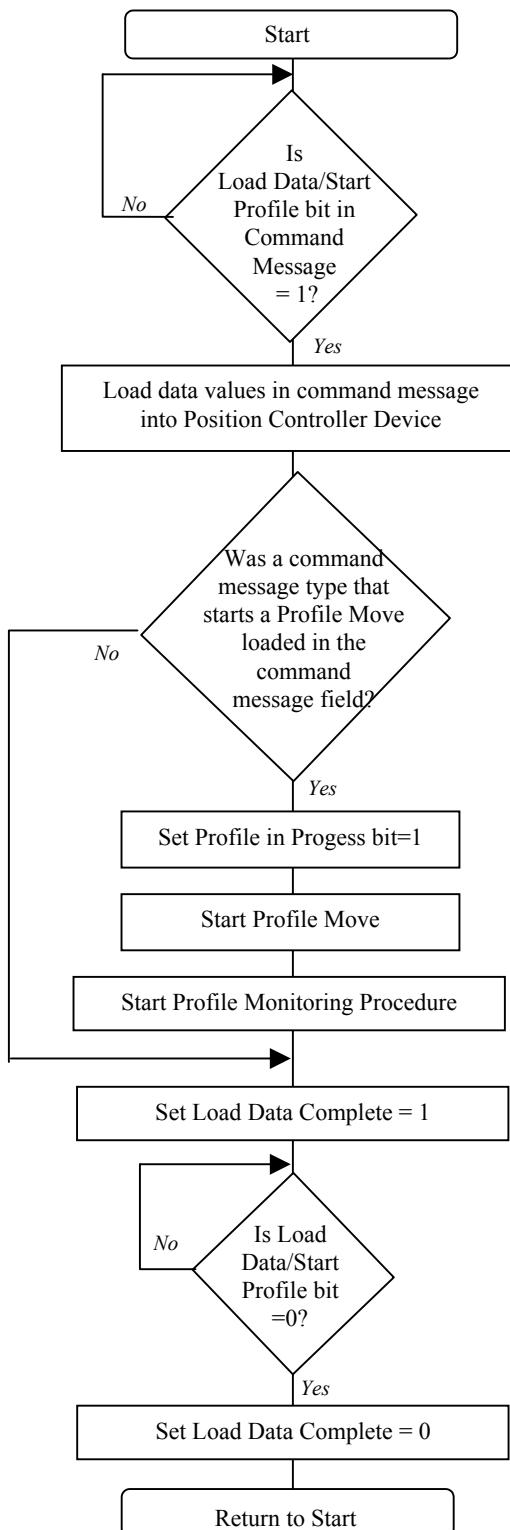
#### 6-18.4.4 I/O Handshaking Procedure

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 Flowcharts A and B below. The behavior required from the server to implement the handshake procedure is described in Flowcharts C and D. Refer to the timing diagram below for representative timing of these bits during the handshake sequence.

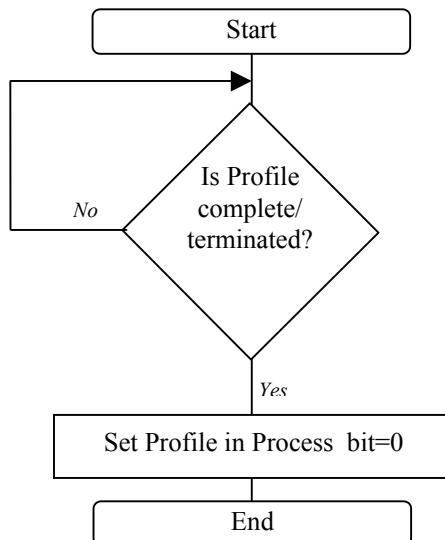
**Flowchart A: Client Data Loading Procedure**



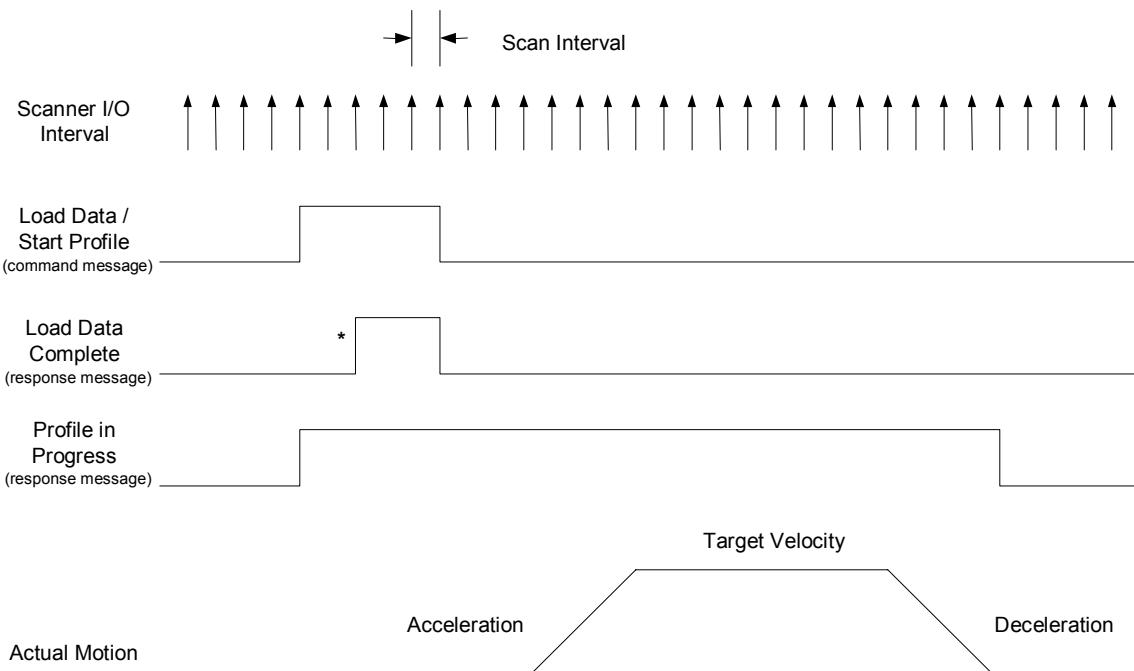
**Flowchart B: Client Profile Move Procedure**

**Flowchart C: Server Behavior**

### Flowchart D: Profile Monitoring Procedure



### 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.

#### 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					
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												

#### 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					
	Enable	Reg Arm	Hard Stop	Smooth Stop	Direction (V. Mode)	Incremental	Start Block	Load Data/ Start Profile					
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												

#### 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					
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												

**Command Message Type 04 hex Deceleration - 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					Deceleration Low Byte			
5					Deceleration Low Middle Byte			
6					Deceleration High Middle Byte			
7					Deceleration High Byte			

**Command Message Type 05 hex Torque - 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					Torque Low Byte			
5					Torque Low Middle Byte			
6					Torque High Middle Byte			
7					Torque High Byte			

**Command Message Type 19 hex Motor Data Attribute - 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					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			

**Command 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 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			

**Command 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 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											

**Command 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 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											

**Command Message Type 1D hex Drive Attribute - 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	Drive Attribute to Get											
2	Command Axis Number				Command Message Type							
3	Drive Attribute to Set											
4	Drive Attribute Value Low Byte											
5	Drive Attribute Value Low Middle Byte											
6	Drive Attribute Value High Middle Byte											
7	Drive Attribute Value High Byte											

**Command Message Type 1E hex Command Block Attribute - 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	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											

**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												

**Semantics:****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.

**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.

**Incremental**

This bit is used to define the position value as either absolute or incremental. 0 = absolute position value and 1 = incremental position value.

**Direction**

This bit is used to control the direction of the motor in Velocity mode. A 1 = forward, positive and a 0 = reverse, negative.

**Smooth Stop**

This bit is used to bring the motor to a controlled stop at the currently implemented deceleration rate.

**Hard Stop**

This bit is used to bring the motor to an immediate stop.

**Arm**

This bit is used to arm the registration input. When the registration input is triggered, the registration action will be executed.

**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.

**Block #**

This byte defines the block number to be executed when the Start Block bit transitions from zero to one.

**Command Message Type**

This field defines the Command Message Type **Response Message Type**

This field defines the Response Message Type

**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:

Command Axis Number	Byte 2		
	Bit 7	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.

**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.

**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

**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..

**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

**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)

**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.

**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.

**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.

**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.

**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.

**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.

**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.

**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.

### 6-18.5.2. 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.

**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								

**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								

**Response Message Type 03 hex Actual Velocity - 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 Velocity Low Byte								
5	Actual Velocity Low Middle Byte								
6	Actual Velocity High Middle Byte								
7	Actual Velocity High Byte								

**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								

**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								

**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								

**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								

**Response Message Type 08 hex Captured Registration 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		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								

**Response Message Type 14 hex Command/Response Error – Required**

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	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								

**Response Message Type 19 hex Motor 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	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								

**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			Response Message Type					
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								

**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			Response Message Type					
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								

**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			Response Message Type					
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								

**Response Message Type 1D hex Drive 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			Response Message Type					
4	Drive Attribute Value Low Byte								
5	Drive Attribute Value Low Middle Byte								
6	Drive Attribute Value High Middle Byte								
7	Drive Attribute Value High Byte								

**Response Message Type 1E hex Command Block 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	Command Block 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	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								

**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	
1	Parameter Instance 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	Parameter Value Low Byte								
5	Parameter Value Low Middle Byte								
6	Parameter Value High Middle Byte								
7	Parameter Value High Byte								

**Semantics:****Profile in Progress**

This bit indicates that a profile move is in progress.

**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.

**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.)

**General Fault**

This bit indicates the logical “or” of all fault conditions.

**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.

**Home Level**

This bit reflects the level of the home input.

**Reg Level**

This bit reflects the level of the registration input.

**Enable**

This bit indicates the state of the enable output. A 1 indicates the enable output is active.

**Executing Block #**

This byte defines the currently executing block if the Block In Execution bit is active.

**Fault Input Fault**

This bit indicates that the fault input is active.

**Fwd Limit**

This bit indicates that the forward input is active.

**Rev Limit**

This bit indicates that the reverse input is active.

**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.

**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.

**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.

**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. **Load Complete**

This bit indicates that the command data contained in the command message has been successfully loaded into the device.

**Response Message Type**

This byte defines the Response Message Type

### 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.

Response Axis Number	Byte 2		
	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.

### 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.

### Commanded Position - Response Message Type 02 hex

This double word reflects the commanded or calculated position in position units.

### Actual Velocity - Response Message Type 03 hex

This double word reflects the actual velocity in profile units.

### Command Velocity - Response Message Type 04 hex

This double word reflects the commanded or calculated velocity in profile units.

### Torque - Response Message Type 05 hex

This double word reflects the torque.

### Home Position - Response Message Type 06 hex

This double word reflects the captured home position in position units.

### Index Position - Response Message Type 07 hex

This double word reflects the captured index position in position units.

### Registration Position - Response Message Type 08 hex

This double word reflects the captured registration position in position units.

### **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 H for a complete list of General Error codes.

General Error Code	Additional Code	Response	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.

### **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.

### **Attribute Value – Response Message Types 19 – 1E hex**

This double word reflects the value of the attribute to get.

### **Object Attribute to Get – Response Message Types 19 – 1E hex**

This byte defines the object attribute from which to get the value.

### **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.

### **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.

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

The following table indicates the I/O Data Attribute mapping for the Position Controller Profile 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

Data Component Name	Class		Instance Number	Attribute		Data Type
	Name	#		Name	#	
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	14	DINT
Actual Velocity	Position Controller	25 <sub>hex</sub>	1-7	Actual Velocity	15	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 Type: 03hex

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.

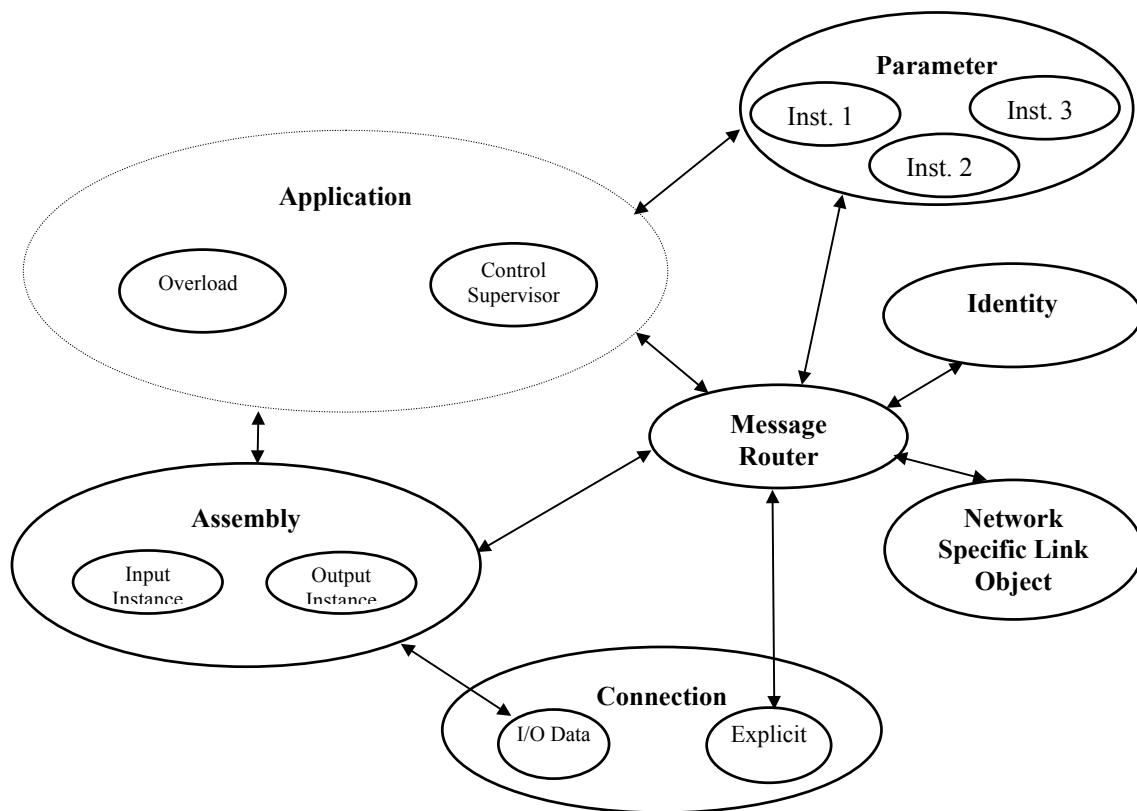
This profile makes Motor Starters of the same device type inter-operable, but not directly interchangeable without doing configuration through a unit’s local interface, a network configuration tool or other means of configuring outside the CIP interface.

### 6-19.1. Object Model

The Object Model in Figure 6-19.1 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

Object Class	Optional/Required	# of Instances
Identity	Required	1
Message Router	Optional	1
Network Specific Link Object	Required	1
Connection	Required	2
Assembly	Optional	1
Parameter	Optional	-
Control Supervisor	Required	1
Overload	Required	-

**Figure 6-19.1. 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.

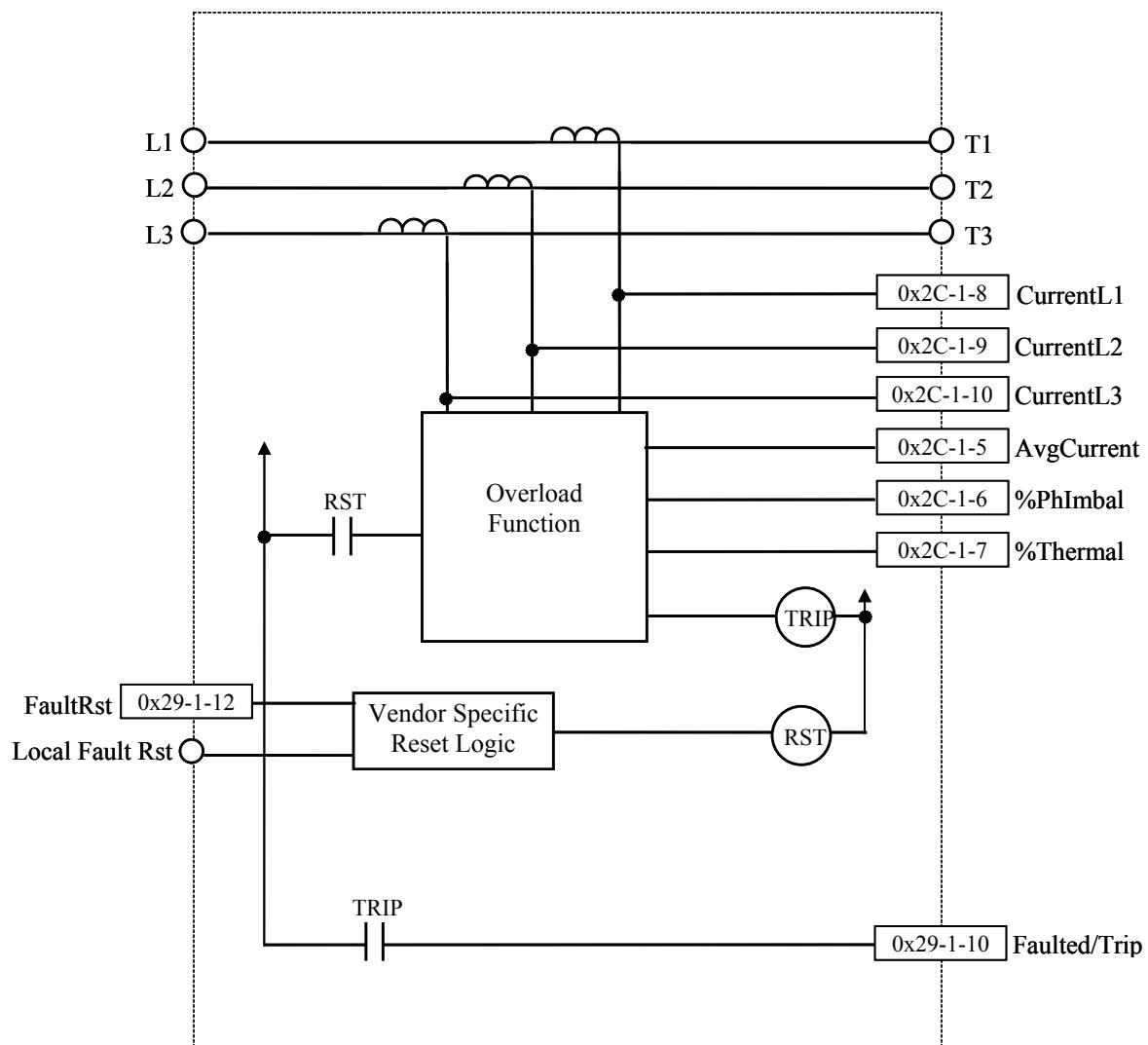
Object	Effect on behavior
Identity	Supports the Reset service
Message Router	No effect
Network Specific Link Object	Configures port attributes
Connection	Logical ports into or out of the device
Assembly	Defines I/O data format
Parameter	Provides a public interface to device configuration data
Control 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:

Object	Interface
Identity	Message Router
Message Router	Explicit Message Connection
Network Specific Link Object	Message Router
Connection	Message Router
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



## 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.

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 Overloads.

Instance	Type	Name
2	Output	Basic Overload
50	Input	Basic Overload
51	Input	Extended Overload

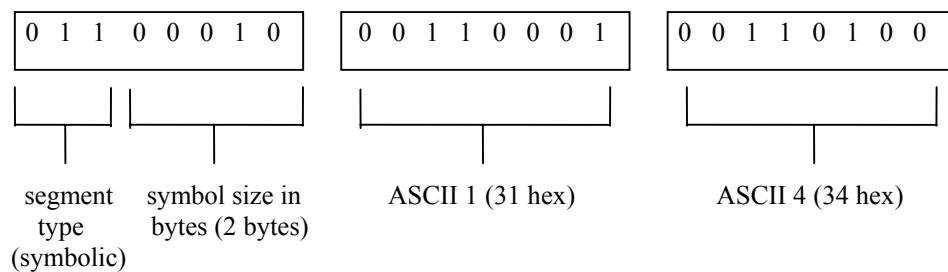
If a bit is not used in an IO Assembly, it is reserved for use in other Assemblies. Reserved bits in Output Assemblies are ignored by the consuming device. Reserved bits in Input Assemblies are set to zero by the producing device.

### 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.

The following example shows the Symbolic Segment used to specify Output Assembly Instance 20 (14 hex).



## 6-19.6. I/O Assembly Data Attribute Format

### 6-19.6.1. Output Assembly Data Attribute Format

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

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						

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

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

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**

The profile for this device type will be defined by the Open DeviceNet Vendor Association, Inc. and ControlNet International.

## 6-21. ENCODER

The profile for this device type will be defined by the Open DeviceNet Vendor Association, Inc. and ControlNet International.

## 6-22. RESOLVER

**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

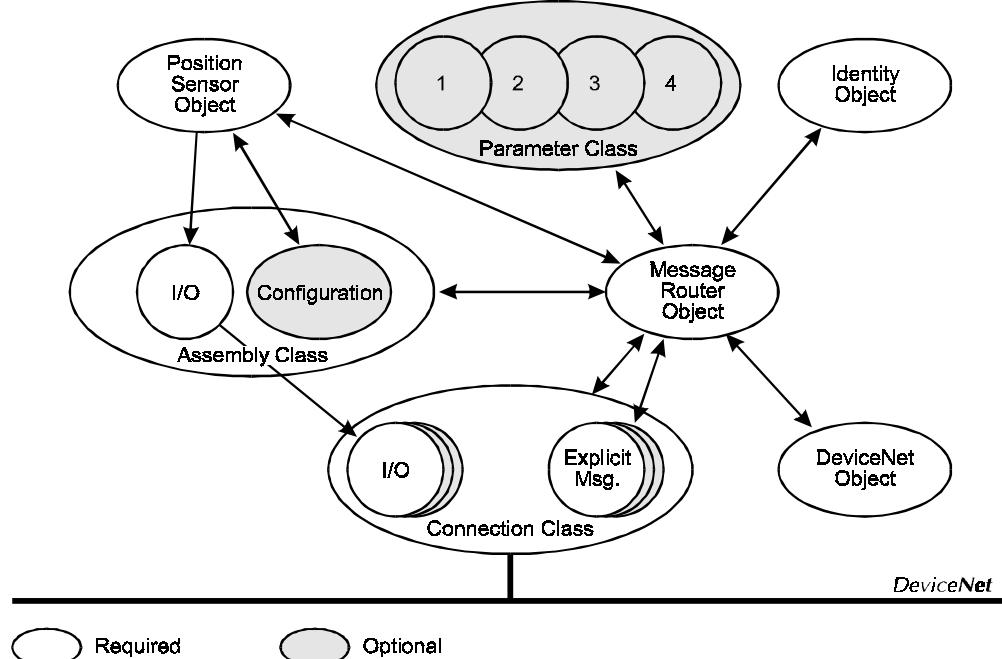
The Object Model in figure 6-22.1 represents a resolver. The table below indicates

- 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.

Object Class	Optional / Required	# of Instances
Identity	Required	1
Message Router	Required	1
Network Specific Link Object	Required	1
Connection	Required	at least 1 I/O and 1 explicit
Assembly	Required	at least 1 I/O input assembly
Parameter	Optional	4
Position Sensor	Required	1

**Figure 6-22.1 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.

Object	Effect on Behavior
Identity	Supports the reset service
Message Router	No effect
Network Specific Link Object	Configures port attributes
Connection	Contains the number of logical ports into or out of the device
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.

Object	Interface
Identity	Message Router
Message Router	Explicit Messaging Connection Instance
Network Specific Link Object	Message Router
Connection	Message Router
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.

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.

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								
		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.

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.

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.

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.

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.

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.

Configuration Parameter	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                               $Max Instances
Descriptor=0x09                           $Parameter Class Descriptor
CfgAssembly=2                            $Configuration Assembly Instance

[Params]
Param1=
    0,                                     $Resolution parameter
    6, "20 23 24 01 30 05",                $Path size and path to attribute
    0x0000,                                $Descriptor
    8, 1,                                    $Data type and size (USINT)
    "Bit Resolution",                      $Name
    "Bits",                                 $Units
    "",                                     $(not used)
    1, 32, (vendor specific),             $Min, max and default values
    0, 0, 0, 0, 0, 0, 0;                  $(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;
                                         $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;
                                         $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;
                                         $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.

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

The profile for this device type will be defined by the Open DeviceNet Vendor Association, Inc. and ControlNet International.

## 6-24. MESSAGE DISPLAY

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 Type: 1B hex**

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 6-26.1 and the object classes are described in the following table:

Object Classes	Class ID	Required / Optional	# of Instances
Identity	0x01	Required	1
Message Router	0x02	Required	1
Network Specific Link Object	0x03	Required	1
Assembly	0x04	Required	1 or more *
Connection	0x05	Required	2 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.

## 6-27. CONTACTOR

Device Type: 15hex

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.

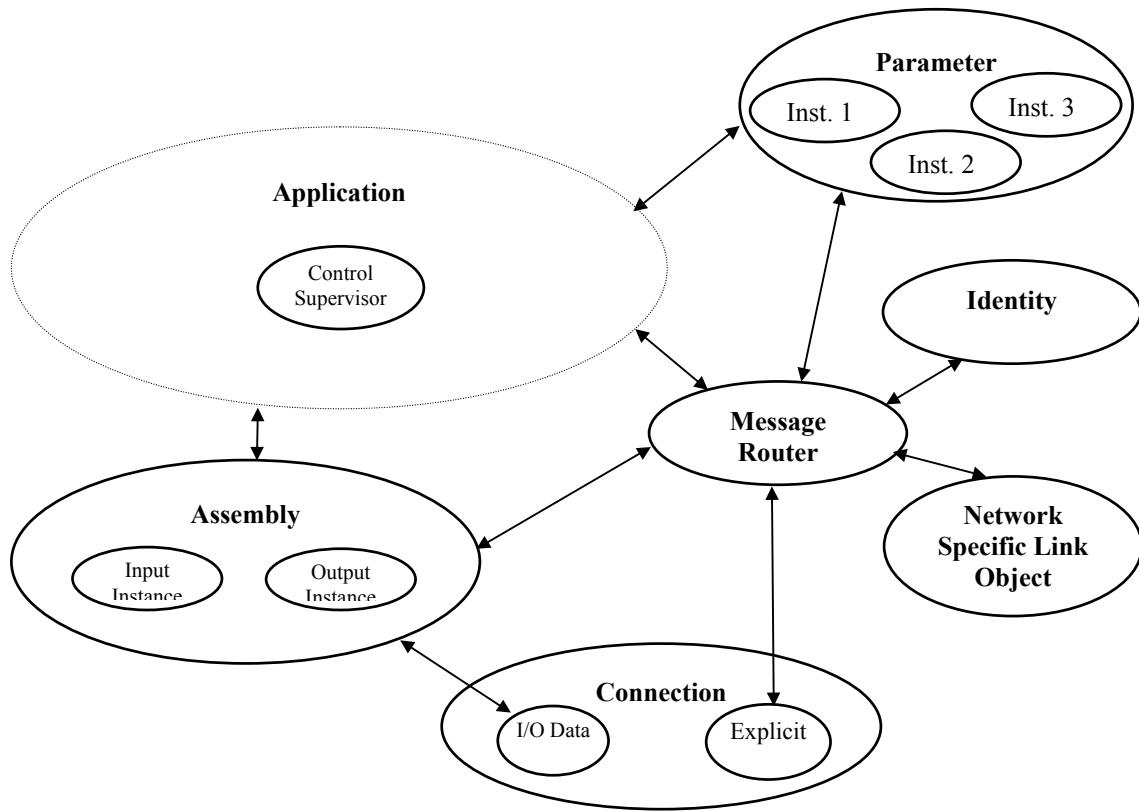
This profile makes Motor Starters of the same device type inter-operable, but not directly interchangeable without doing configuration through a unit’s local interface, a network configuration tool or other means of configuring outside the CIP interface.

### 6-27.1. Object Model

The Object Model in Figure 6-27.1 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

Object Class	Optional/Required	# of Instances
Identity	Required	1
Message Router	Optional	-
Network Specific Link Object	Required	1
Connection	Required	2
Assembly	Optional	1
Parameter	Optional	-
Control Supervisor	Required	1

**Figure 6-27.1** 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.

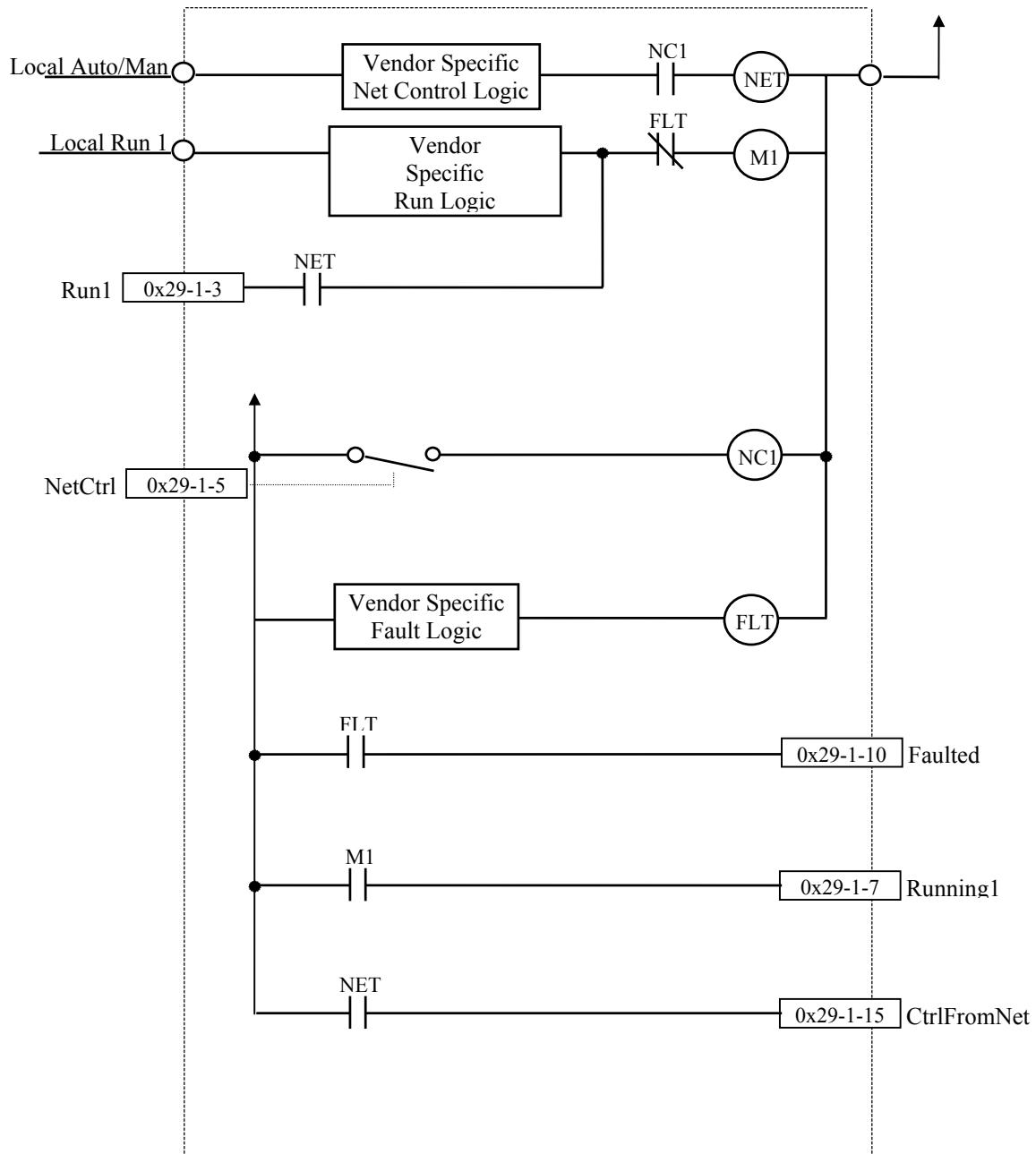
Object	Effect on behavior
Identity	Supports the Reset service
Message Router	No effect
Network Specific Link Object	Configures port attributes
Connection	Logical ports into or out of the device
Assembly	Defines I/O data format
Parameter	Provides a public interface to device configuration data
Control 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:

Object	Interface
Identity	Message Router
Message Router	Explicit Message Connection
Network Specific Link Object	Message Router
Connection	Message Router
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



## 6-27.5. 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.

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.

Instance	Type	Name
1	Output	Basic Contactor
4	Output	Extended Contactor

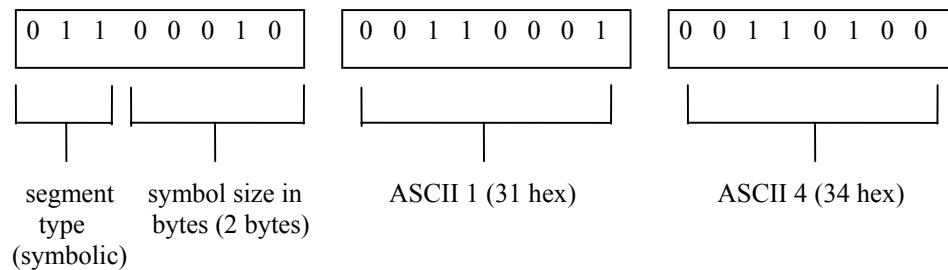
If a bit is not used in an IO Assembly, it is reserved for use in other Assemblies. Reserved bits in Output Assemblies are ignored by the consuming device. Reserved bits in Input Assemblies are set to zero by the producing device.

### 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).



## 6-27.6. I/O Assembly Data Attribute Format

### 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						

### 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.

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 Type: 16hex

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.

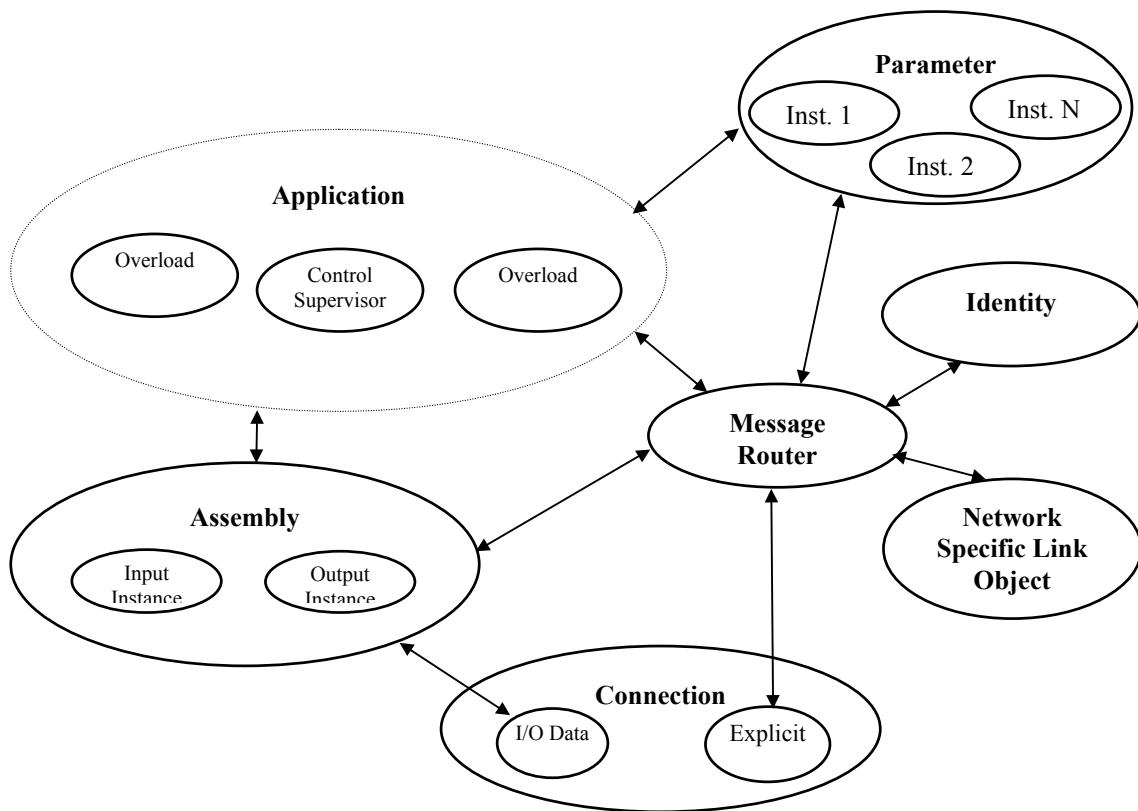
This profile makes Motor Starters of the same device type inter-operable, but not directly interchangeable without doing configuration through a unit’s local interface, a network configuration tool or other means of configuring outside the CIP interface.

### 6-28.1. Object Model

The Object Model in Figure 6-28.1. 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

Object Class	Optional/Required	# of Instances
Identity	Required	1
Message Router	Optional	1
Network Specific Link Object	Required	1
Connection	Required	2
Assembly	Optional	1
Parameter	Optional	-
Control Supervisor	Required	1
Overload	Required	-

**Figure 6-28.1. 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.

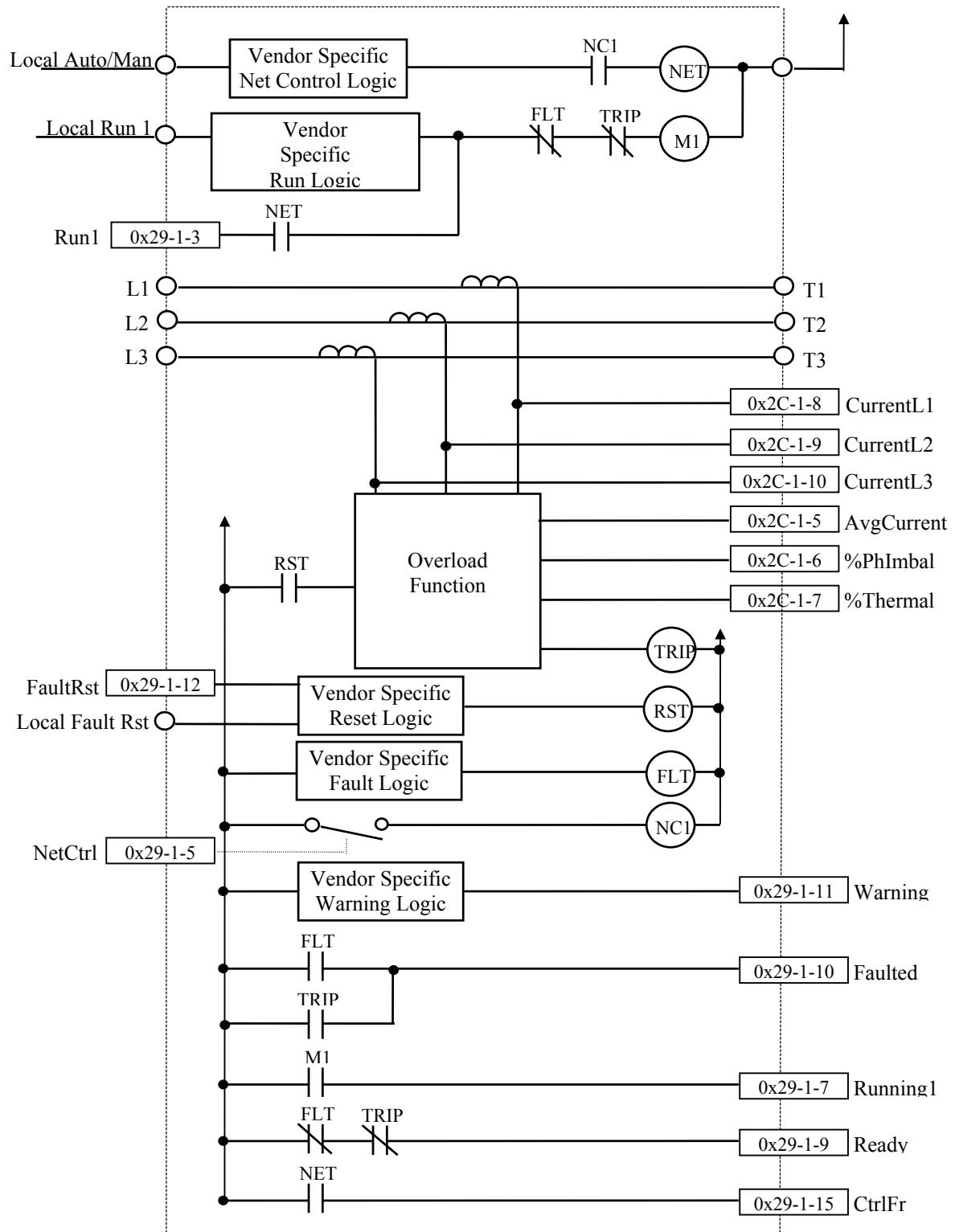
Object	Effect on behavior
Identity	Supports the Reset service
Message Router	No effect
Network Specific Link Object	Configures port attributes
Connection	Logical ports into or out of the device
Assembly	Defines I/O data format
Parameter	Provides a public interface to device configuration data
Control Supervisor	Manages motor functions and operational states
Overload	Implements overload

### 6-28.3. Defining Object Interfaces

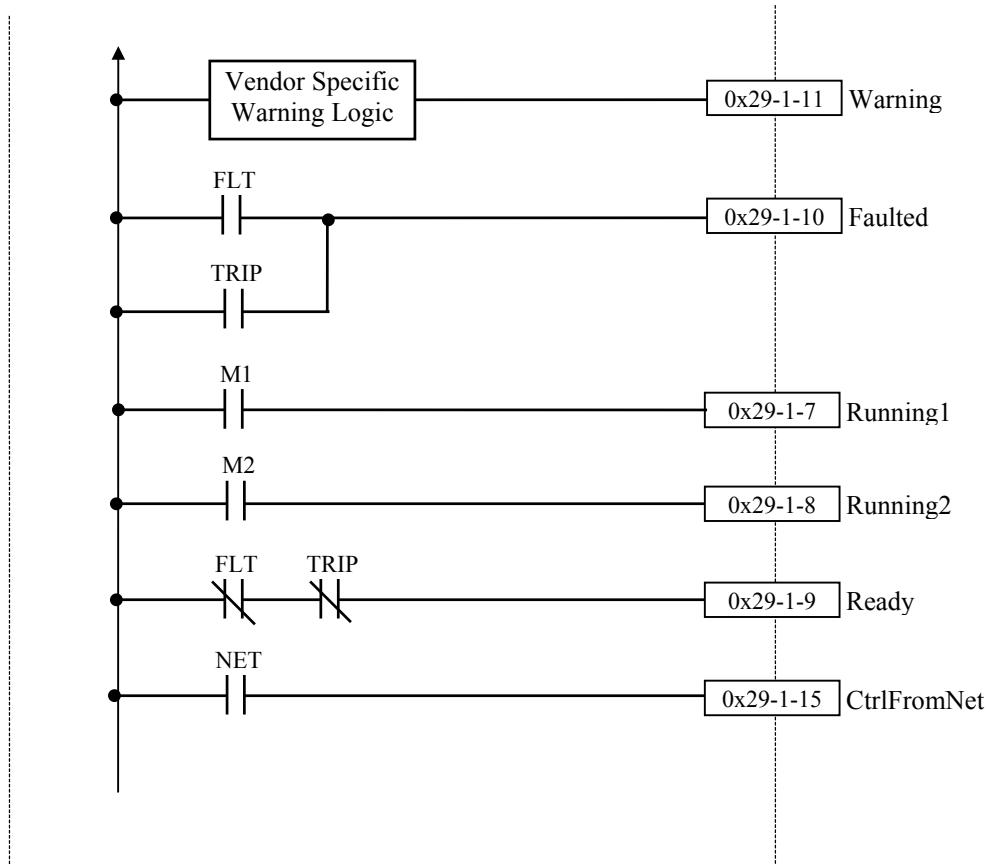
The objects in the Motor Overload have the interfaces listed in the following table:

Object	Interface
Identity	Message Router
Message Router	Explicit Message Connection
Network Specific Link Object	Message Router
Connection	Message Router
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

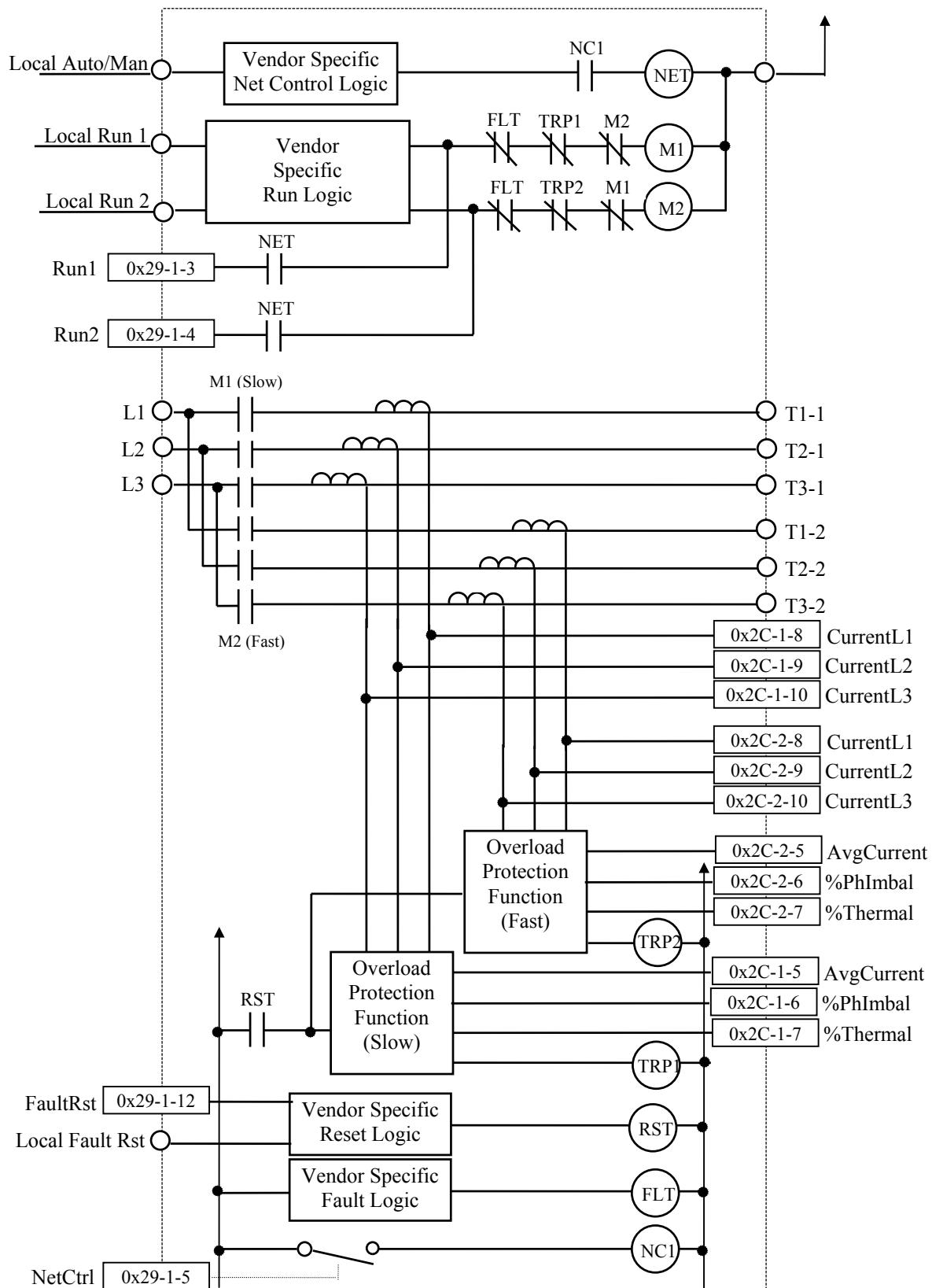
### 6-28.3.1 Starter Interface and Behavior

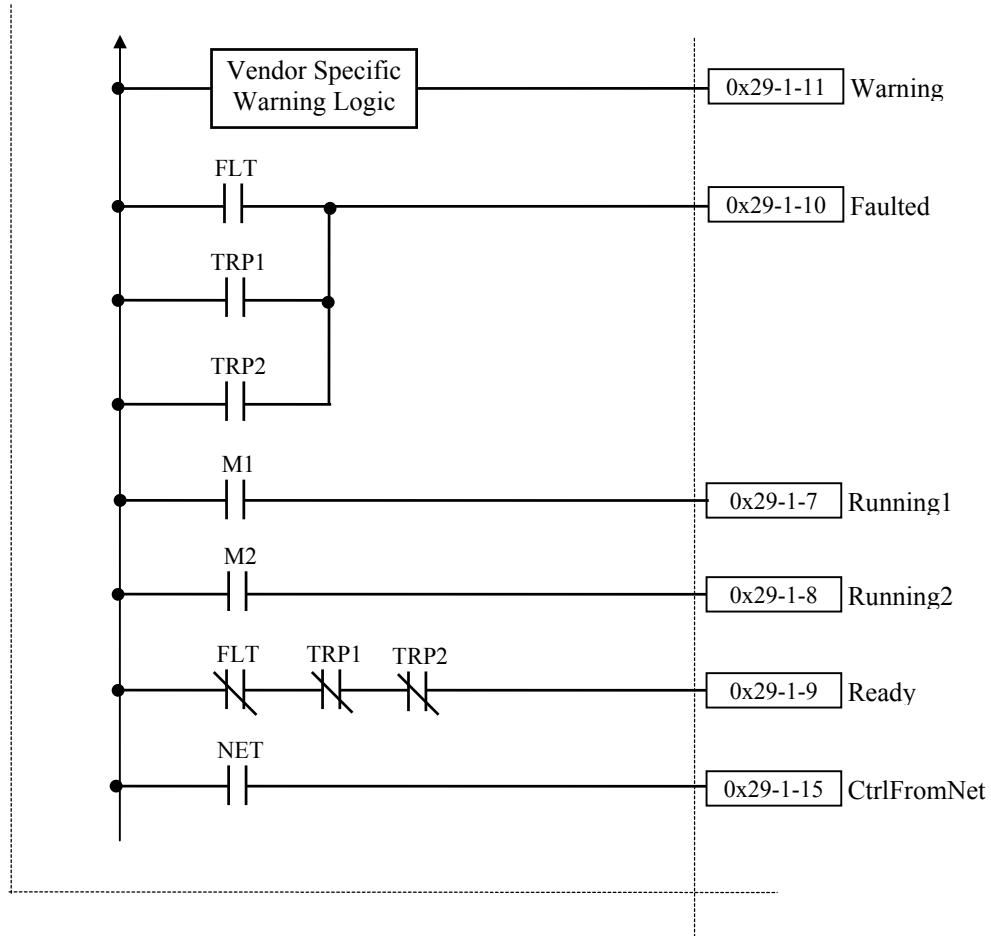


### 6-28.3.2 Reversing Motor Starter Interface and Behavior



### 6-28.3.3 Two Speed Motor Starter Interface and Behavior





#### 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.

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.

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

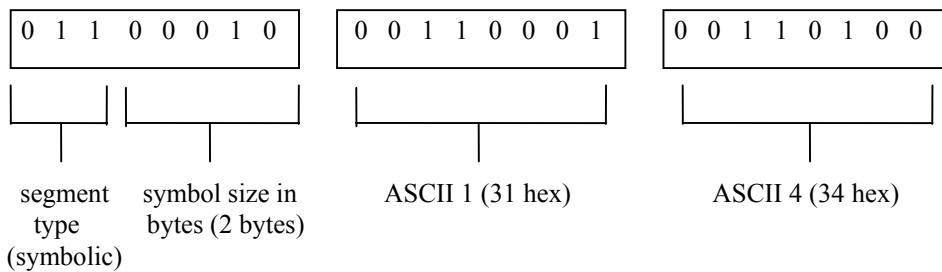
If a bit is not used in an IO Assembly, it is reserved for use in other Assemblies. Reserved bits in Output Assemblies are ignored by the consuming device. Reserved bits in Input Assemblies are set to zero by the producing device.

#### 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).



## 6-28.5. I/O Assembly Data Attribute Format

### 6-28.5.1 Output Assembly Data Attribute Format

#### 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

#### 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

#### 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

#### 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

#### 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

#### 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

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

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 Type: 17hex

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.

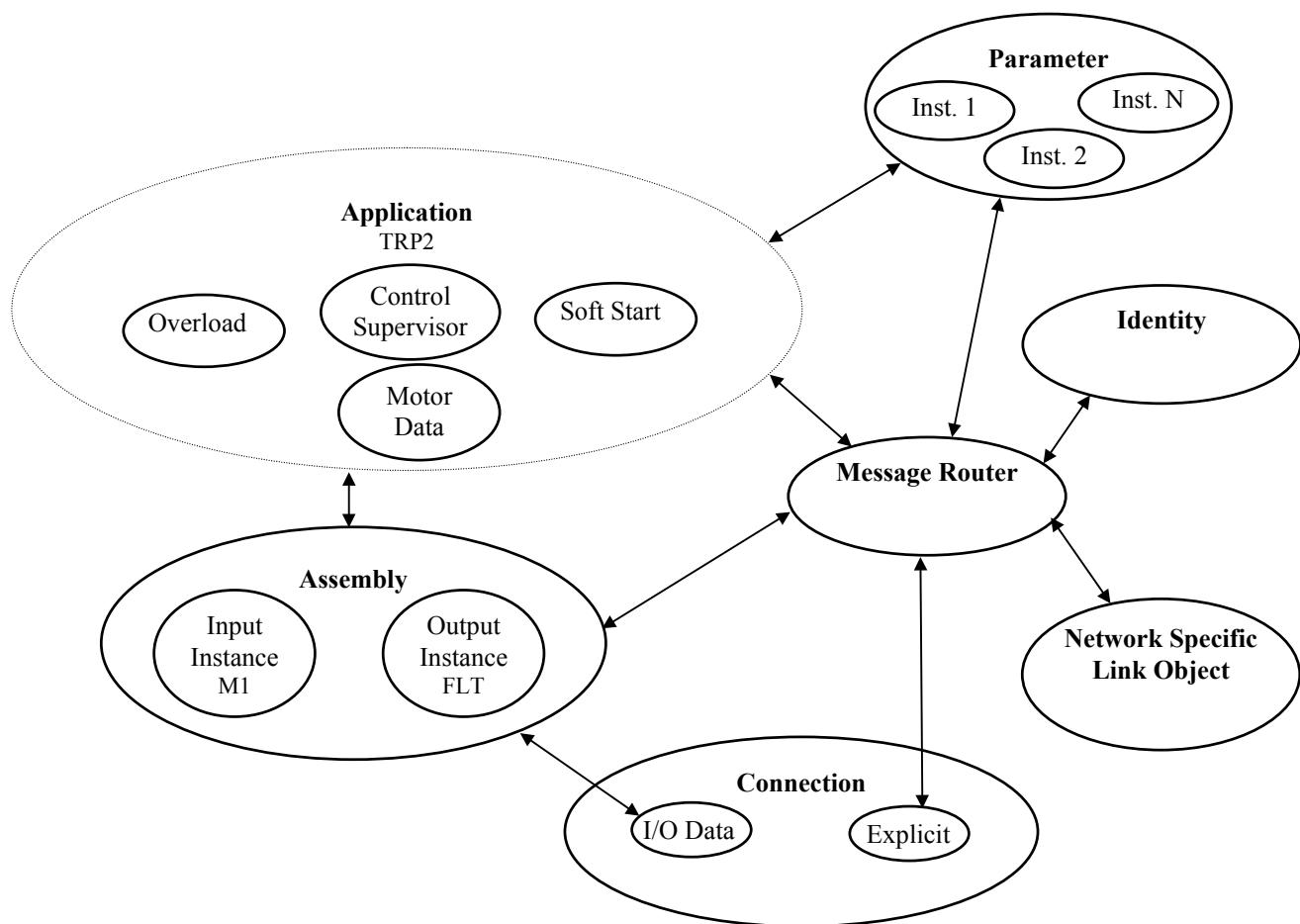
This profile makes Softstart Starters of the same device type inter-operable, but not directly interchangeable without doing configuration through a unit’s local interface, a network configuration tool or other means of configuring outside the CIP interface.

### 6-29.1. Object Model

The Object Model in Figure 6-29.1 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

Object Class	Optional/Required	# of Instances
Identity	Required	1
Message Router	Optional	1
Network Specific Link Object	Required	1
Connection	Required	2
Assembly	Optional	1
Parameter	Optional	-
Control Supervisor	Required	1
Softstart	Optional	-
Overload	Optional	-
Motor Data	Optional	-

**Figure 6-29.1. 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.

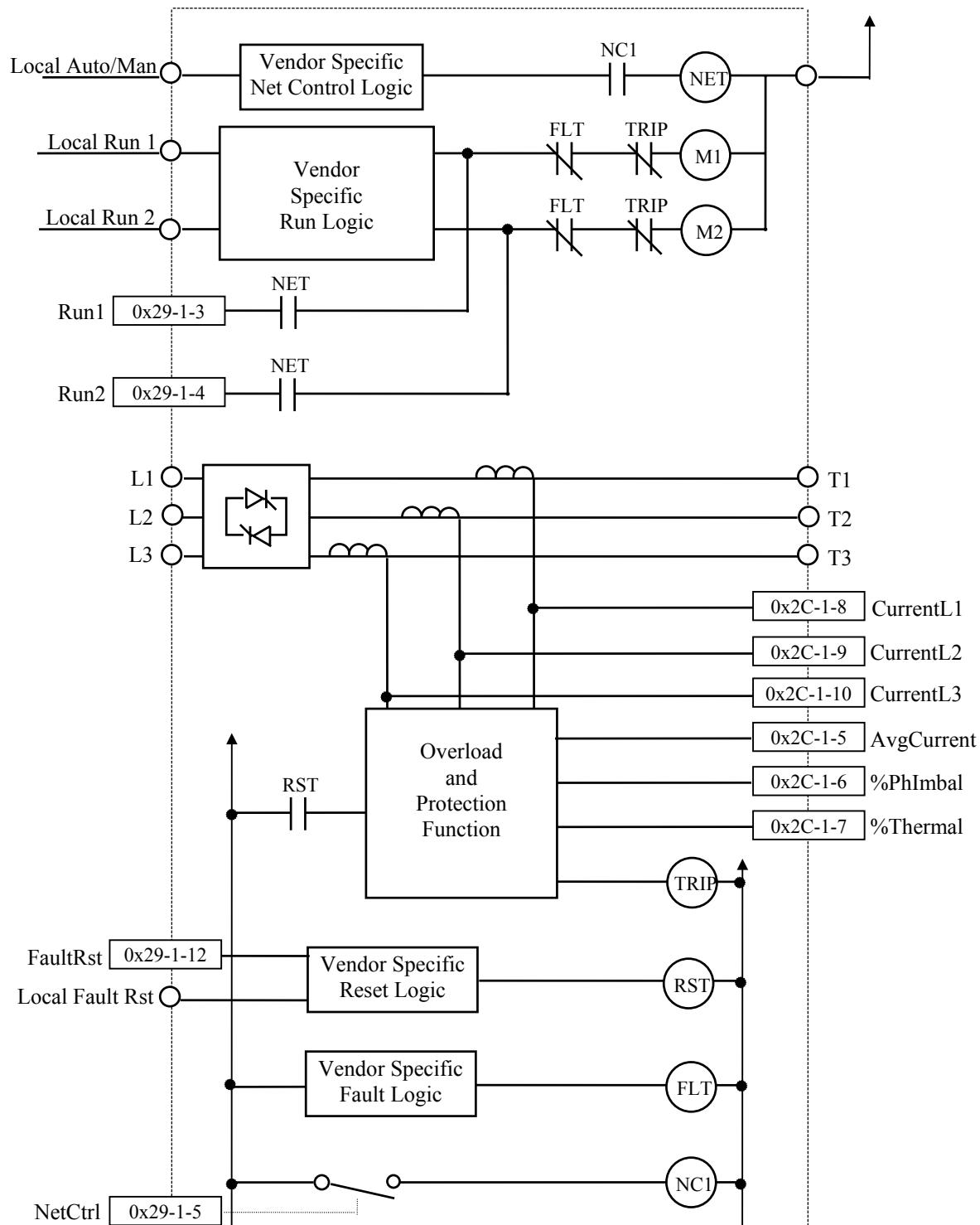
Object	Effect on behavior
Identity	Supports the Reset service
Message Router	No effect
Network Specific Link Object	Configures port attributes
Connection	Logical ports into or out of the device
Assembly	Defines I/O data format
Parameter	Provides a public interface to device configuration data
Control 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.

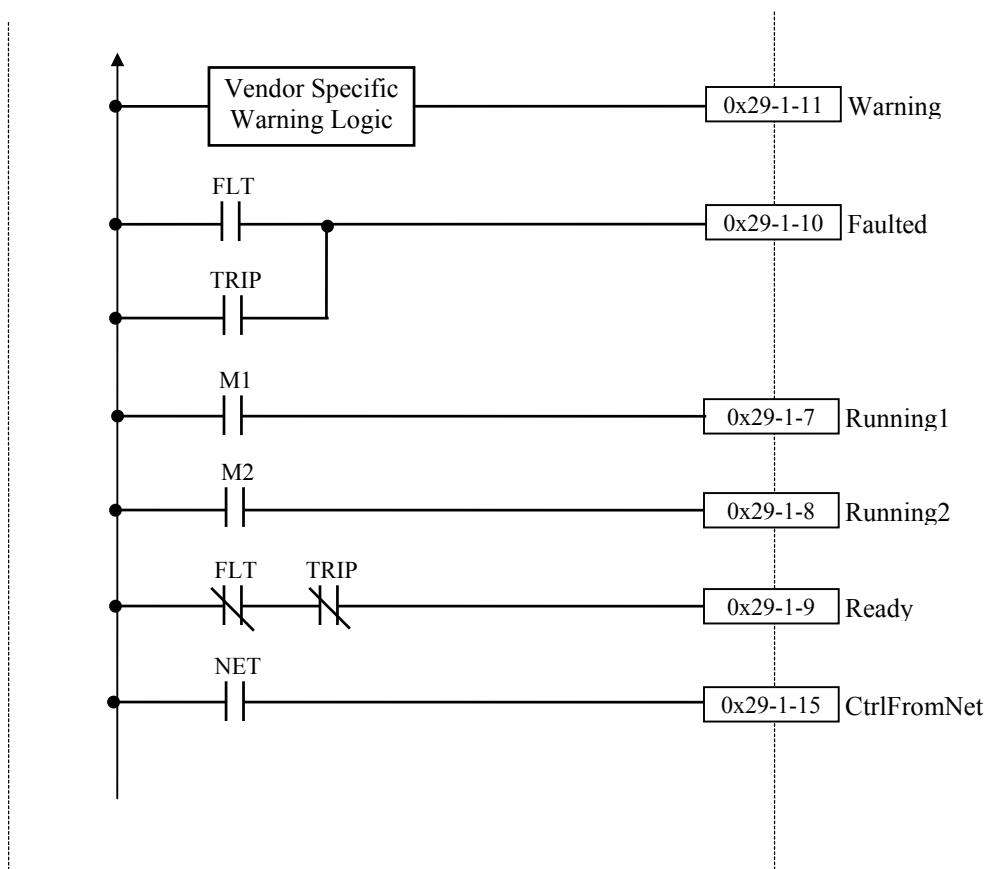
## 6-29.3. Defining Object Interfaces

The objects in the Motor Overload have the interfaces listed in the following table:

Object	Interface
Identity	Message Router
Message Router	Explicit Message Connection
Network Specific Link Object	Message Router
Connection	Message Router
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





## 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.

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.

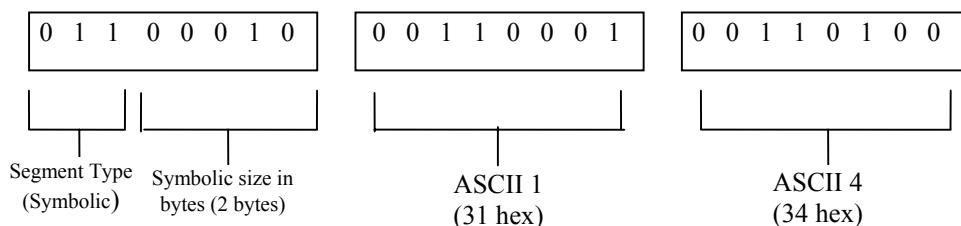
Instance	Type	Name
60	Input	Basic SoftStart
61	Input	Extended SoftStart

### 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).



## 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

#### 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

#### 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

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	2D	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: 18hex

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.

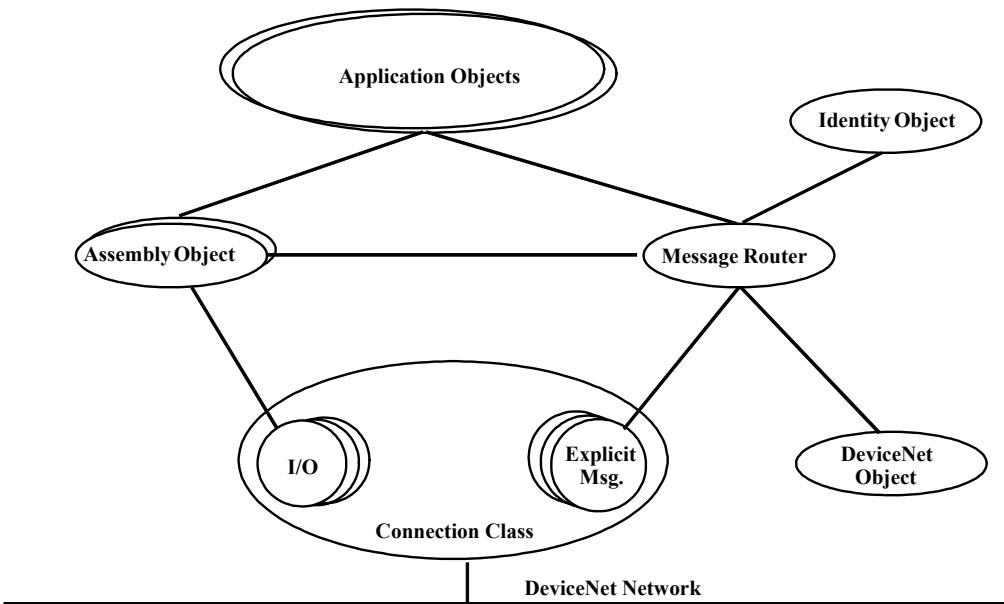
### 6-30.1. Object Model

The Object Model in Figure 6-30.1. 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

Object Class	Optional/Required	# of Instances
Identity	Required	at least 1
Message Router	Required	1
Network Specific Link Object	Required	at least 1
Connection	Required	at least 1 I/O and 1 explicit
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 Chapter 2, Contents of a Device Profile.

**Figure 6-30.1. 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.

Object	Effect on behavior
Identity	Supports the Reset service
Message Router	No effect
Network Specific Link Object	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
Application	Defines device operation

## 6-30.3. Defining Object Interfaces

The objects in the HMI Device have the interfaces listed in the following table:

Object	Interface
Identity	Message Router
Message Router	Explicit Messaging Connection Instance
Network Specific Link Object	Message Router
Connection Class	Message Router
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 6-31.1. 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

Object Class	Optional/Required	# of Instances
Identity	Required	1
Message Router	Required	1
Network Specific Link Object	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-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).

### 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.

## 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.

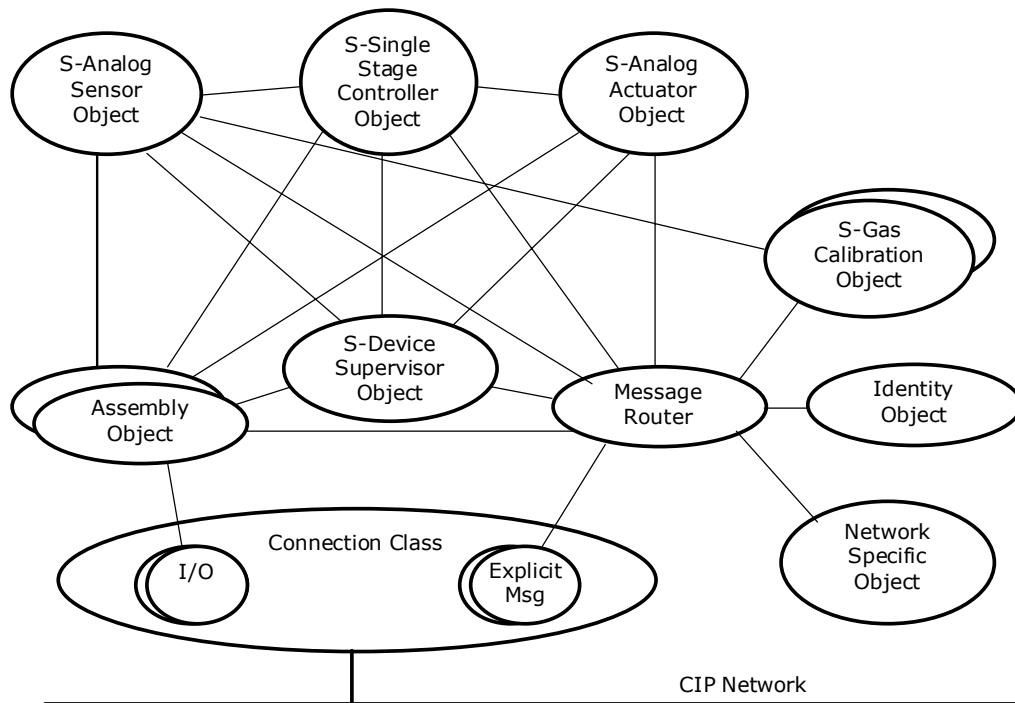
**S-Analog Sensor Object Subclasses**

Instance ID	Subclass Name	Subclass ID (Attribute 99 Value)	Required	Function	Restrictions
1	Flow Diagnostics	01	Required	Added diagnostics for MFC	None

**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 6-31.1. Object Model for the MFC Device**



### 6-31.2. How Objects Affect 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
Network Specific Link Object	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 CIP specific objects (i.e., Identity, Network Specific Link Object, 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

### 6-31.3. Defining Object Interfaces

Object	Interface
Identity	Message Router
Message Router	Explicit Messaging Connection Instance
Network Specific Link Object	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

### 6-31.4. I/O Assembly Instances

The following table identifies the I/O assembly instances supported by the MFC.

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

Number	Required	Type	Name
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.

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)							

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	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)							
8	0	Override							
	1	Setpoint (low byte)							
	2	Setpoint (high byte)							
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)							

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	14	Exception Detail Warning 6 (manufacturer, 0)							
13	0	Flow (low byte)							
	1	Flow							
	2	Flow							
	3	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)							
17	0	Status							
	1	Flow (low byte)							
	2	Flow							
	3	Flow							
	4	Flow (high byte)							
	5	Setpoint (low byte)							
	6	Setpoint							

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	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)							

### 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.

Data Component	Class	Instance	Attribute
----------------	-------	----------	-----------

Name	Name	Number	Number	Name	Number	Type
Flow	S-Analog Sensor	31hex	1	Indicated Flow	6	INT
Valve	S-Analog Actuator	32hex	1	Value	6	INT
Override	S-Analog Actuator	32hex	1	Override	5	USINT
Setpoint	S-Single Stage Controller	33hex	1	Setpoint	6	INT
Status	S-Device Supervisor	30hex	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	31hex	1	Indicated Flow	6	REAL
FP-Valve	S-Analog Actuator	32hex	1	Value	6	REAL
FP-Setpoint	S-Single Stage Controller	33hex	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

##### Limitations

Attribute	Limitation
Device Type	Supported Values: “MFC” = Mass Flow Controller device “MFM” = Mass Flow Meter device

##### 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.

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	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 *
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

#### Limitations

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 K)	Supported Values = {Counts; scem}	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

#### 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 K)	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

##### 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 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.

#### 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.

## 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 6-32.1 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

Object Class	Optional/Required	# of Instances
Identity	Required	1
Message Router	Required	1
DeviceNet	Required	1
Connection	Required	At least 1 I/O polled and 1 Explicit.
Assembly	Required	At least 1 Input
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

## 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.

### S-Analog Sensor Class Level Object Subclasses

Instance ID	Subclass Name	Subclass ID (Attribute 99 Value)	Required	Function	Restrictions
Class	Instance Selector	01	Required	Data Flow from the Active Instance	None

## Instance Subclasses

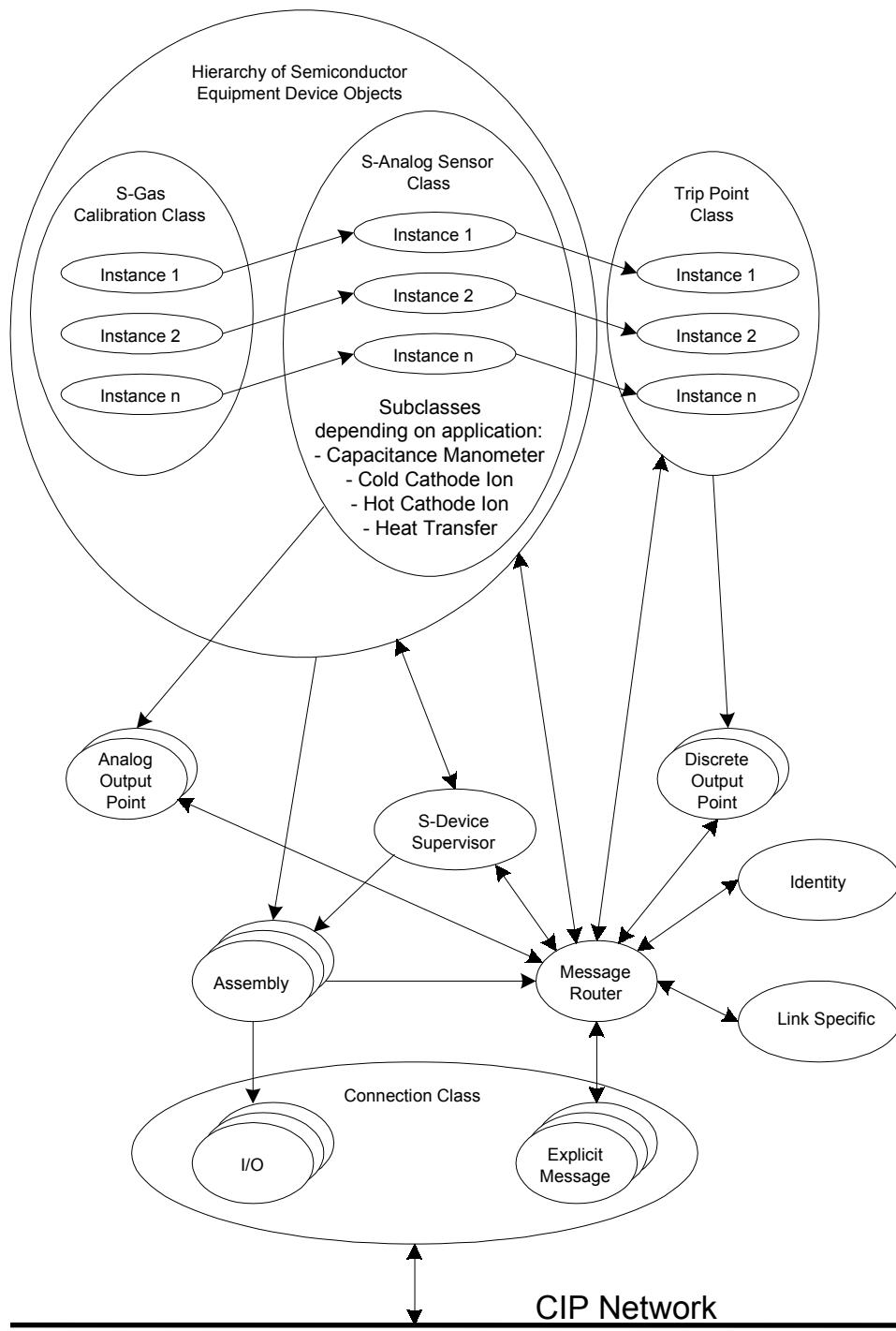
Each instance level subclass defines a unique meaning for an overlapping range of instance attribute IDs and/or instance service IDs. The range for subclass definitions begins at ID 96 and numbers downward for attributes, and ID 63<sub>hex</sub> and numbers downward for services. The subclass for a given 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.

### 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 6-32.1. Object Model.**

### 6-32.2. How Objects Affect 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
Link Specific	Configures port attributes
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	<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 DeviceNet objects (i.e., Identity, DeviceNet, Connection, etc.).</p>
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.

### 6-32.3. Defining Object Interfaces

Object	Interface
Identity	Message Router
Message Router	Explicit Messaging Connection Instance
Link Specific	Message Router
Connection Class	Message Router
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

### 6-32.4. 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.

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	1	Trip Status	Trip Status	Trip Status	Trip Status	Trip Status	Trip Status	Trip Status	
				8	Inst. 8	Inst. 7	Inst. 6	Inst. 5	Inst. 4	Inst. 3	Inst. 2	
			2 - 3	Exception Status								
				1	Trip Status	Trip Status	Trip Status	Trip Status	Trip Status	Trip Status	Trip Status	
				8	Inst. 8	Inst. 7	Inst. 6	Inst. 5	Inst. 4	Inst. 3	Inst. 2	
				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	1	Trip Status	Trip Status	Trip Status	Trip Status	Trip Status	Trip Status	Trip Status	
				8	Inst. 8	Inst. 7	Inst. 6	Inst. 5	Inst. 4	Inst. 3	Inst. 2	
			2 - 5	Exception Status								
				1	Trip Status	Trip Status	Trip Status	Trip Status	Trip Status	Trip Status	Trip Status	
				8	Inst. 8	Inst. 7	Inst. 6	Inst. 5	Inst. 4	Inst. 3	Inst. 2	
				2 - 5	REAL Pressure Value							
7	N	Input	0	1	Trip Status	Trip Status	Trip Status	Trip Status	Trip Status	Trip Status	Trip Status	
				8	Inst. 8	Inst. 7	Inst. 6	Inst. 5	Inst. 4	Inst. 3	Inst. 2	
			2 - 5	Exception Status								
8	N	Input	0	Exception Status								
9	N	Input	0 - 1	Active Instance								
			2 - 3	INT Active Pressure Value								
10	N	Input	0	1 - 2	Exception Status							
			3 - 4	Active Instance								
			2 - 3	INT Active Pressure Value								
11	N	Input	0	1	Trip Status	Trip Status	Trip Status	Trip Status	Trip Status	Trip Status	Trip Status	
				8	Inst. 8	Inst. 7	Inst. 6	Inst. 5	Inst. 4	Inst. 3	Inst. 2	
			2 - 3	Exception Status								
				1	Trip Status	Trip Status	Trip Status	Trip Status	Trip Status	Trip Status	Trip Status	
				8	Inst. 8	Inst. 7	Inst. 6	Inst. 5	Inst. 4	Inst. 3	Inst. 2	
				2 - 3	Active Instance							

Instance	Required	Type	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0			
			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										
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										

Instance	Required	Type	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
			10 - 13 14 - 17								REAL Pressure Value 3 REAL Pressure Value 4
21	N	Input	0 - 1 2 - 3 4 - 5 6 - 7 8 - 9 10 - 11 12 - 13 14 - 15								INT Pressure Value 1 INT Pressure Value 2 INT Pressure Value 3 INT Pressure Value 4 INT Pressure Value 5 INT Pressure Value 6  INT Pressure Value 7  INT Pressure Value 8
22	N	Input	0 1 - 2 3 - 4 5 - 6 7 - 8 9 - 10 11 - 12 13 - 14 15 - 16								Exception Status INT Pressure Value 1 INT Pressure Value 2 INT Pressure Value 3 INT Pressure Value 4 INT Pressure Value 5 INT Pressure Value 6  INT Pressure Value 7  INT Pressure Value 8
23	N	Input	0 1 2 - 3 4 - 5 6 - 7 8 - 9 10 - 11 12 - 13 14 - 15 16 - 17	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	Exception Status INT Pressure Value 1 INT Pressure Value 2 INT Pressure Value 3 INT Pressure Value 4 INT Pressure Value 5  INT Pressure Value 6  INT Pressure Value 7  INT Pressure Value 8
24	N	Input	0 - 3 4 - 7 8 - 11 12 - 15 16 - 19								REAL Pressure Value 1 REAL Pressure Value 2 REAL Pressure Value 3 REAL Pressure Value 4  REAL Pressure Value 5

Instance	Required	Type	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
			20 - 23 24 - 27 28 - 31								REAL Pressure Value 6
											REAL Pressure Value 7
											REAL Pressure Value 8
25	N	Input	0 1 - 4 5 - 8 9 - 12 13 - 16 17 - 20 21 - 24 25 - 28 29 - 32								Exception Status REAL Pressure Value 1 REAL Pressure Value 2 REAL Pressure Value 3 REAL Pressure Value 4  REAL Pressure Value 5  REAL Pressure Value 6  REAL Pressure Value 7  REAL Pressure Value 8
26	N	Input	0 1 Trip Status Inst. 8 2 - 5 6 - 9 10 - 13 14 - 17 18 - 21 22 - 25 26 - 29 30 - 33	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	Exception Status  REAL Pressure Value 1 REAL Pressure Value 2 REAL Pressure Value 3  REAL Pressure Value 4  REAL Pressure Value 5  REAL Pressure Value 6  REAL Pressure Value 7  REAL Pressure Value 8

All of the elemental components listed above follow standard CIP Data Encoding as specified in Appendix C.

#### 6-32.4.1. 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. See text of 6-32.4.1.

The following table indicates the I/O assembly Data attribute mapping for this Gauge device.

Data Component Name	Class	Class Number	Instance Number	Attribute		
				Name	Number	Type
Pressure Value	S-Analog Sensor	31hex	1 – N	Value	6	INT or REAL
Trip Status	Trip Point	35hex	1 - N	Status	7	BOOL
Exception Status	S-Device Supervisor	30hex	1	Exception Status	12	BYTE
Active Instance	S-Analog Sensor	31hex	Class Level	Active Instance Number	95	UINT
Active Pressure Value *	S-Analog Sensor	31hex	Class Level	Active Value	94	INT or REAL

\* For Combination Gauges only

## 6-32.5. Object Limitations and Specific Definitions

### 6-32.5.1. S-Device Supervisor Object Instance

#### 6-32.5.1.1. 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.

Exception Detail Warning (Attribute 14):

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 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							

Combination and Multi-Gauge Devices Only:

Device N Exception Detail (Warning) Byte 0	S-Analog Sensor object Instance N — Status Extension							
Device N Exception Detail (Warning) Byte 1	S-Analog Sensor object Instance N — Sensor Warning — Byte 0							
Device N Exception Detail (Warning) Byte 2	S-Analog Sensor object Instance N — Sensor Warning — Byte 1							

[End Combination and Multi-Gauge Devices Only]

Manufacturer Exception Detail (Warning) Size	0	0	0	0	0	0	0	1
Manufacturer Exception Detail (Warning)	8 bits defined by Manufacturer							

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							
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							

### 6-32.5.1.2. Manufacturer's Device Type Attribute Definition

The following limitations apply:

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)

### 6-32.5.1.3. 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).

### 6-32.5.2. S-Analog Sensor Object Instance

#### 6-32.5.2.1. Limitations

##### Object Instance Number Assignment

The following instance numbering limitations apply:

Instance	Description
1 ... 20	Reserved for Pressure Gauges
21 ... 60	Reserved for Interlock-Relay Setting Monitors

##### Data Type, Data Units and Produce Trigger Delta Type

The following attribute limitations apply:

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 Counts.	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.

##### 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.

### 6-32.5.2.2. Object Extensions

This profile uses the extensions for the following 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

### 6-32.5.3. Trip Point Object Instance

#### 6-32.5.3.1. Limitations

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.

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:



Where x is the Instance ID of the S-Analog Sensor object whose *Value* attribute is the source.

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 source attribute uses the following encoding:

24 <sub>hex</sub>	x
-------------------	---

Where x is the Instance ID of the Discrete Output Point object whose *Value* attribute is the Destination.

### 6-32.6. 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. CONTROLNET PROGAMMABLE LOGIC CONTROLLER

**Device Type: 0E<sub>hex</sub>**

The ControlNet Programmable Logic Controller Device type defines a device that acts as a scheduled connection originator. 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

Object Class	Optional/Required	# of Instances
Identity	Required	1
Message Router	Required	1
ControlNet	Required	1
Connection Manager	Required	1
Scheduling	Required	1

The ControlNet 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 ControlNet Programmable Logic Controller Device have the interfaces listed in the following table:

Object	Interface
Identity	Message Router
Message Router	Explicit Messaging Connection Instance
Network Specific Link Object	Message Router
Connection Manager	Message Router
Scheduling	Message Router

## 6-34. CONTROLNET PHYSICAL LAYER COMPONENT

**Device Type: 32<sub>hex</sub>**

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.

## **Volume 1: CIP Common Specification**

### **Chapter 7: Electronic Data Sheets**

---

This page is intentionally left blank

## 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:

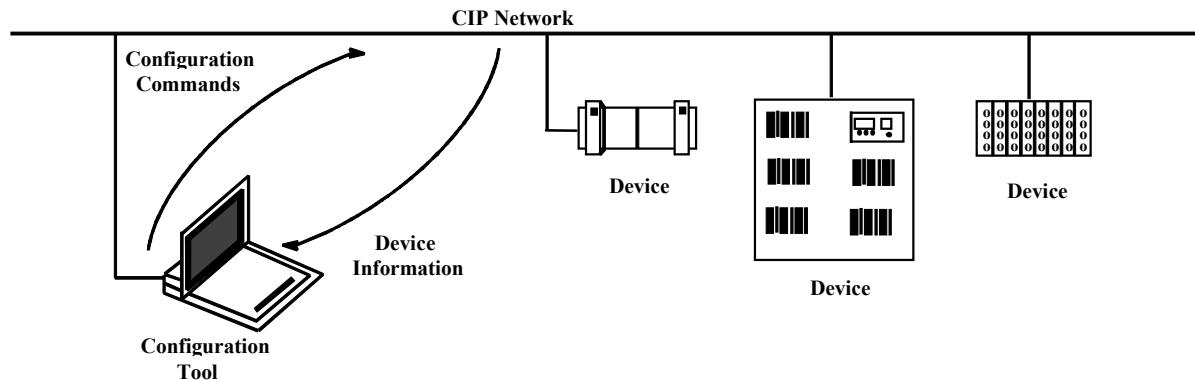
- product developers
- configuration tool designers

### 7-1.1. Terms and Definitions Used in this Chapter

Term	Meaning
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 format	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
parameter object:	An object that has two types, as specified below
full	An object within a device that contains the configuration data value, prompt strings, data conversion factors, and other information associated with a device
stub	A shorthand form of a parameter object that stores only the configuration data value and provides only a standard access point for the parameter

## 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.



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.

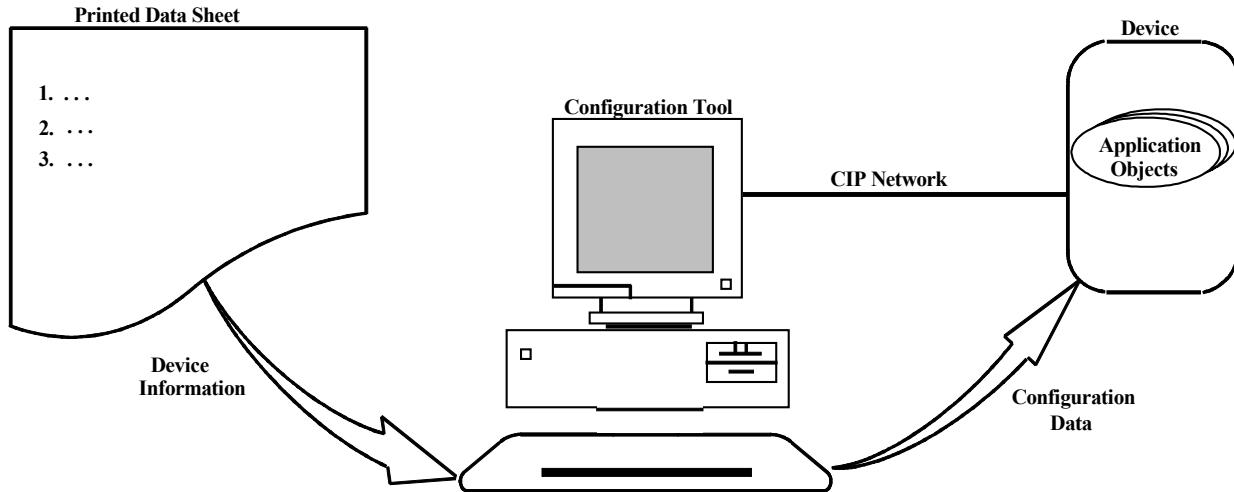
### 7-2.1. Configuration Data Sources and Methods

This chapter 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

#### 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.



### 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:

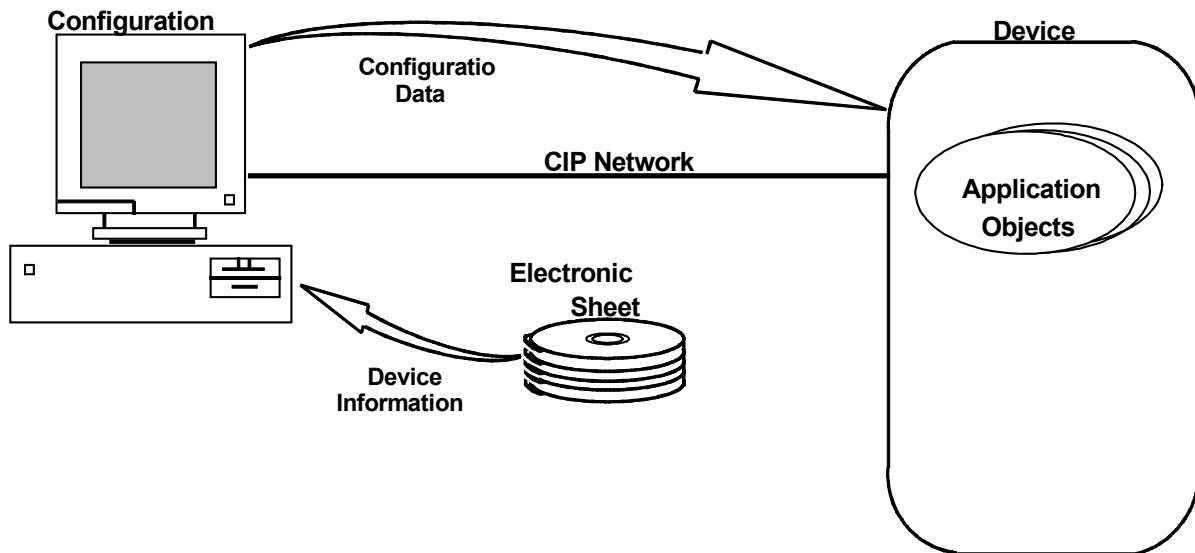
- context
- content
- format

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 4-3 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.



### 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.

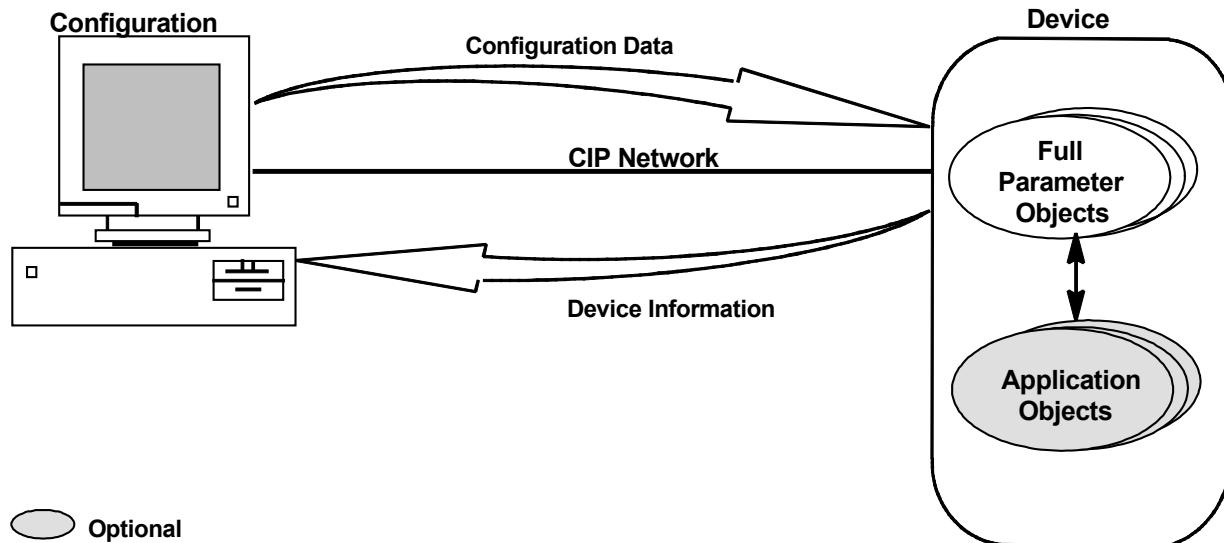
#### Using Full Parameter Objects

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.



### Using Parameter Object Stubs

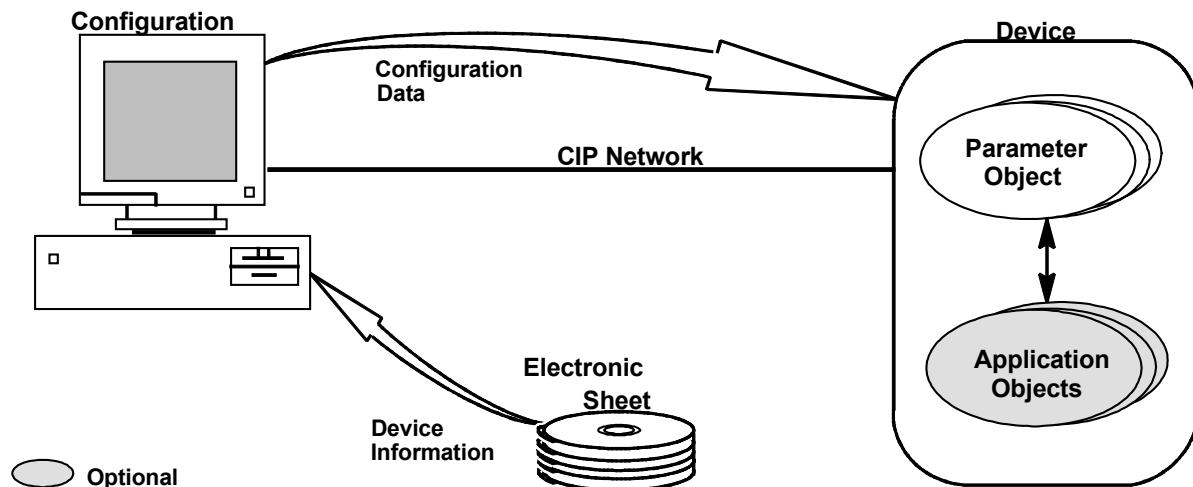
Parameter Object stubs provide an established address to a device's configuration data values without requiring specification of descriptive text, data limits, and other parameter properties. When a device contains Parameter Object stubs, a configuration tool may obtain additional configuration information 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.



### 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 must 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 must:

- 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:

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 must 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:

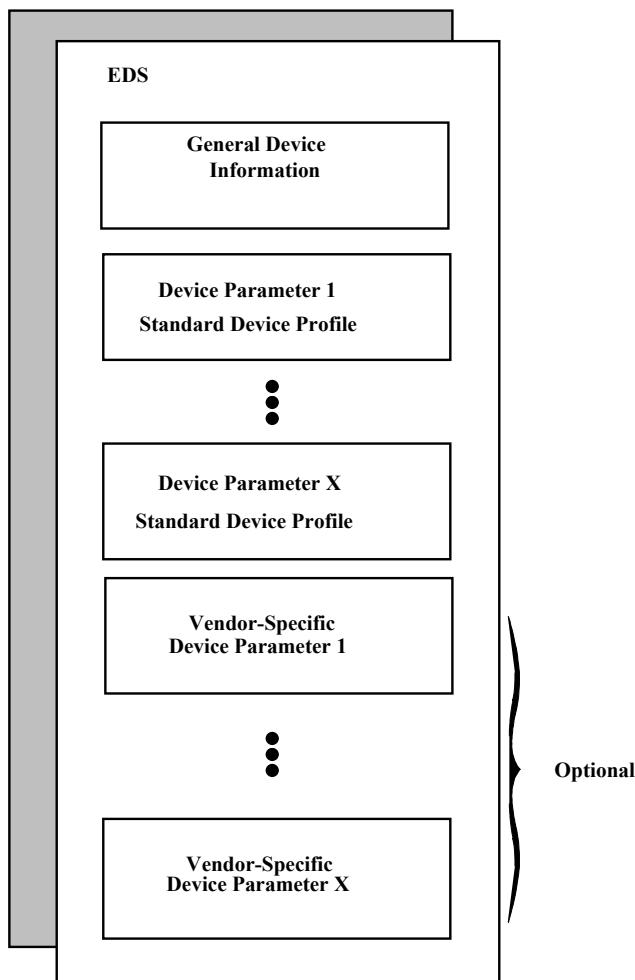
<b>Data Component Name</b>	<b>Class</b>		<b>Instance Number</b>	<b>Attribute</b>		<b>Data Type</b>
	<b>Name</b>	<b>Number</b>		<b>Name</b>	<b>Number</b>	
Output Range	Analog Output	0B <sub>hex</sub>	1	Output Range	7	UINT
		o o o				
Input Range	Analog Input	0A <sub>hex</sub>	1	Input Range	7	BYTE

## 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.

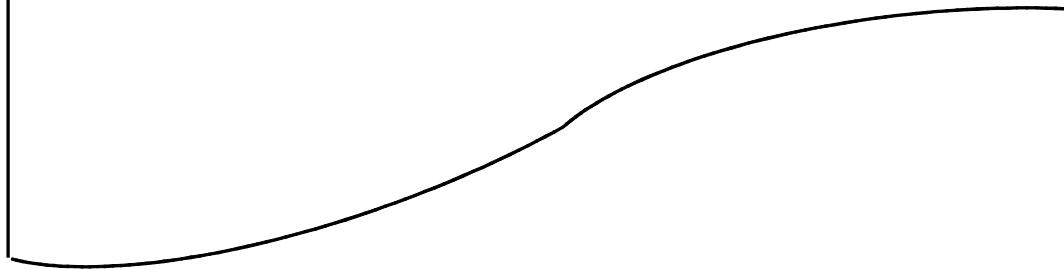
### 7-3.1. The Electronic Data Sheet

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.



### 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.



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		
<b>User Settings</b>	<b>Minimum</b>	<b>Maximum</b>	<b>Factory Default</b>	<b>Access Point</b>
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.

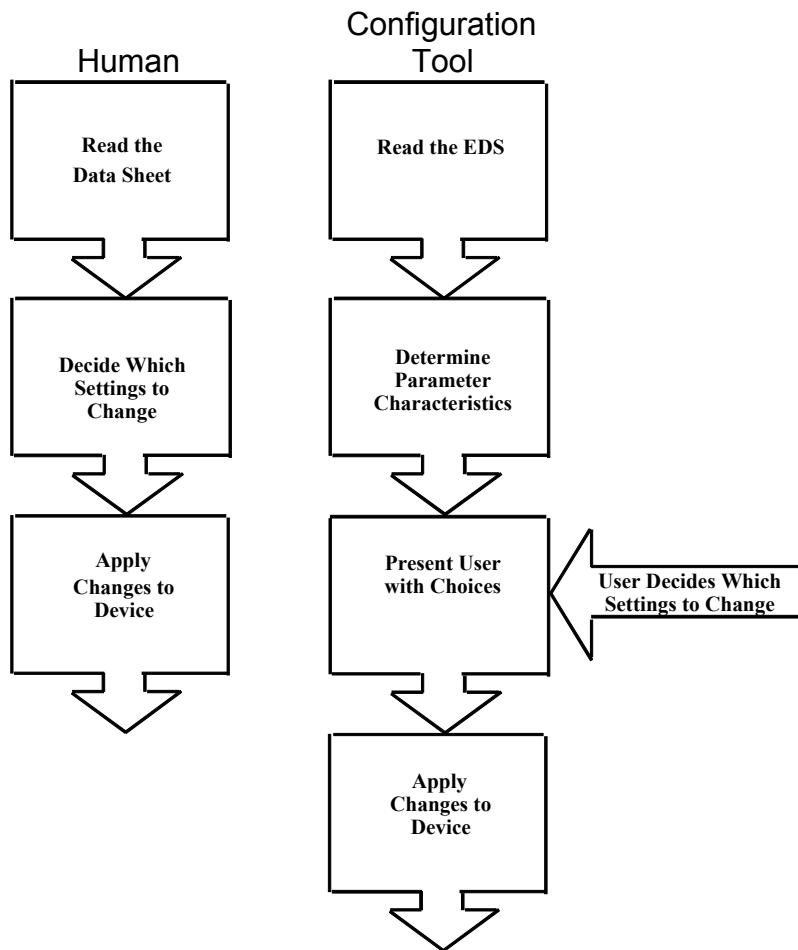
To perform the actual configuration, a configuration tool uses CIP messaging to perform the changes in the device. Currently, the textual information in an EDS must be ASCII-representable characters. Future revisions of this specification will support non-ASCII character representations such as UNICODE.

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



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:

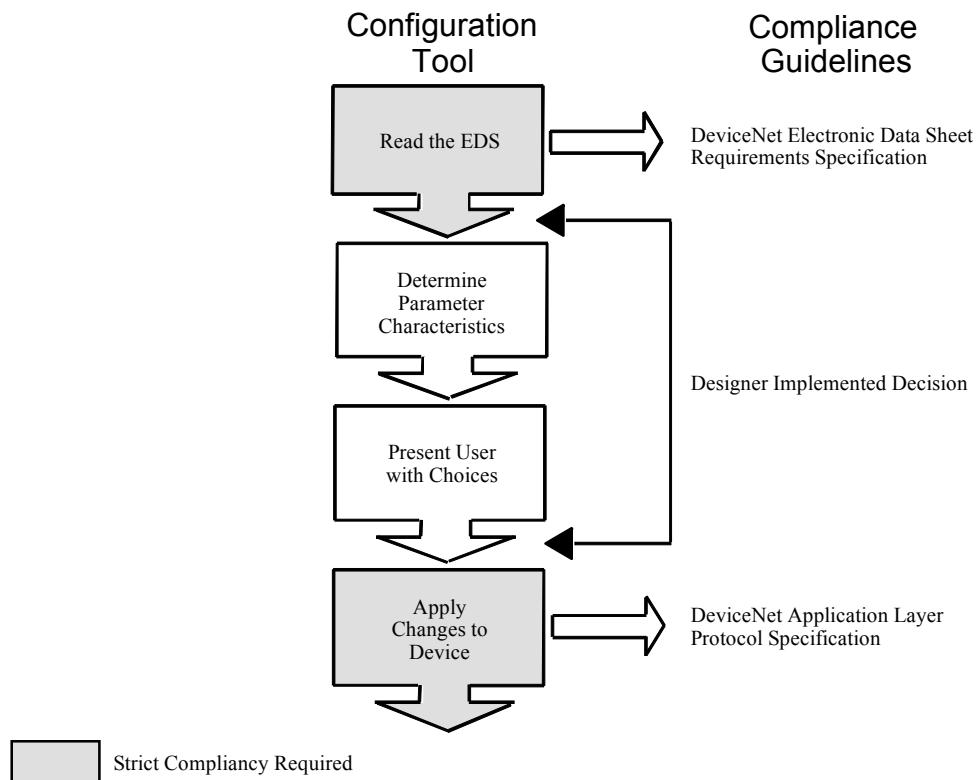
- Consistency among vendors for the benefit of device makers
- Compatibility among vendors for the benefit of tool makers

All EDS developers must implement EDSs compliant with these requirements. Product developers determine all other implementation details. Every EDS interpreter designed for CIP products must 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 must be satisfied for all configuration tools that use an EDS. These requirements describe:

- EDS encoding
- EDS command language
- EDS syntax



### 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

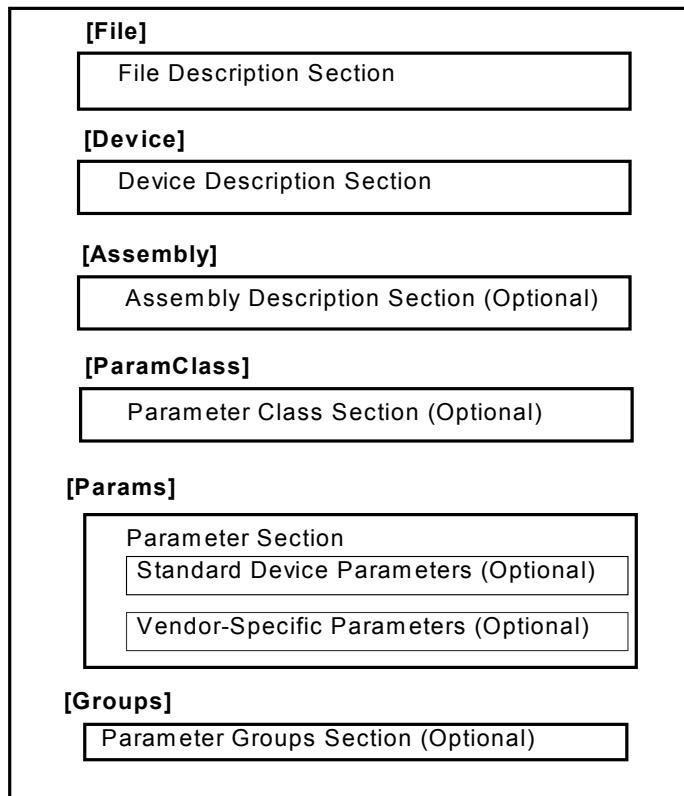
The interpreter must:

- 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 figure below shows a block diagram of the structure of an electronic data sheet. 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.

### Electronic Data Sheet



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.

#### 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.

Section Name	Legal Delimiter	Placement	Required/Optional
File Description	[File]	1	Required
Device Description	[Device]	2	Required
Parameter Class	[ParamClass]	*	Optional
Parameters	[Params]	*	Optional
Parameter Groups	[Groups]	*	Optional
Assembly	[Assembly]	*	Optional
Device Classification	[Device Classification]	*	Optional
Connection Characteristics	[Connection Manager]	*	Optional
Port	[Port]	*	Optional
Modular	[Modular]	*	Optional

\*Placement of optional groups only needs to follow the required groups

Observe these rules when defining an EDS:

- **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* in square brackets (the Legal Delimiter).
- **Section order** - Each required section must be placed in the required order. Optional sections can be omitted entirely or included with empty data place holders.
- **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 fields** - Each entry contains one or more fields. A comma delimiter separates each field. The meaning of the field(s) depends on the context of the section.
- **Vendor-Specific Keywords** - Section and Entry Keywords may be vendor-specific. These keywords shall begin with the Vendor ID of the company making the addition followed by an underscore (*VendorID\_VendorSpecificKeyword*). 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.)

```
[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
```

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.

#### 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.

## 7-3.4. EDS Data Encoding Requirements

This section describes the data encoding requirements for the EDS file.

### 7-3.4.1. ASCII Character 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. Characters that cannot be displayed on an ANSI terminal cannot be used in identifier names or in data representations.

### 7-3.4.2. ASCII 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.

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".**

### 7-3.4.3. 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" "123" "XYZ"	\$this is a comment
-------------------------	---------------------

are also interpreted as

"ABC123XYZ"

#### 7-3.4.4. String Escape Sequences

The interpreter recognizes the following escape characters in a string and translate them:

- \\" translates to a \
- \n translates to a newline
- \t translates to a tab
- \\" translates to a "

#### 7-3.4.5. 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 whitespace 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

#### 7-3.4.6. 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 whitespace 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	0x7FFFFFFF

The range of legal **LINT** data is:

Decimal Notation:	-9223372036854775808	to	223372036854775807
Hexadecimal Notation:	0x8000000000000000	to	0x7FFFFFFFFFFFFF

#### 7-3.4.7. 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 whitespace 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	0b111111111111111111111111111111

The range of legal **LWORD** data is:

Decimal Notation:	0	to	18446744073709551615
Hexadecimal Notation:	0x0	to	0xFFFFFFFFFFFFFF
Binary Notation:	0b00000000000000000000000000000000	to	0b111111111111111111111111111111

#### 7-3.4.8. 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"
```

### 7-3.4.9. EDS White Space

The EDS interpreter treats 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 include:

- New line
- Carriage Return
- Line feed
- Tabs, vertical and horizontal
- End of File marker

### 7-3.4.10. 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:

<pre>\$ This is a valid comment line 1, 2, 3; &lt;NL&gt; \$ Comments cannot span &lt;NL&gt; more than one line &lt;NL&gt;</pre>	<pre>&lt;NL&gt; \$ This is a valid comment &lt;= This is an error - no \$</pre>
---	---

### 7-3.4.11. 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 1996 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 96 through 99 (Note that a leading 19 is implied)

NOTE: Two character year representations are not recommended.

### 7-3.4.12. 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

### 7-3.4.14. ASCII Floating Point Convention (REAL, LREAL)

The REAL and LREAL data types represent binary floating point values. The internal representation of these data formats are 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 value 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  $\pm 9999999999999999$

Scientific Notation: 0.0 to  $\pm nnnn.nnnnnnnnnnE\pm xxxx$

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

#### **7-3.4.15. 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

## 7-3.5. Basic EDS File Requirements

This section describes each section of an EDS and specifies the usage requirements.

For EDS requirements for:	Go to section:
File description section	7-3.5.1
Device description section	7-3.5.2
Device Classification	7-3.5.3
Parameter class section	7-3.5.4
Parameters section	7-3.5.5
Parameter groups section	7-3.5.6
Assembly section	7-3.5.7

### 7-3.5.1. File Description Section

The file description section 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:

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

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
  - 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
- **Home URL** - Uniform Resource Locator of the master EDS file, the Icon file and other files related to this EDS. The HomeURL 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.

For example:      2.1      <=major revision is 2, and minor revision is 1

```
$ Sample Electronic Data Sheet illustrating the File Section

[File]
DescText = "Smart Widget EDS File";
CreateDate = 04-03-94;                                $ created
CreateTime = 17:51:44;
ModDate = 04-06-94;                                    $ last changed
ModTime = 22:07:30;
Revision = 2.1;                                         $ Revision of EDS
HomeURL = "http://www.odva.org/EDS/example.eds";

[Device]
...
[ParamClass]
...
[Params]
...
[Groups]
...
```

### 7-3.5.2. Device Description Section

The Device Description section 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.

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	VendorName	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

<sup>1</sup> The SerNum and Comment entry keywords have been obsoleted

<sup>2</sup> This entry represents an attribute of the Identity Object

<sup>3</sup> 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 Section 3-1. The string for vendor specific device types are 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 contain 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.

```
$ 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]
...
```

### 7-3.5.3. Device Classification

The Device Classification section 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:

- ControlNet;
- DeviceNet;
- EtherNetIP
- 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.

#### 7-3.5.4. Parameter Class Section

The parameter class section:

- identifies the class level attributes of the configuration parameters described by the EDS
- contains a subset of the Parameter Object class attributes as defined in the CIP Object Library

The following table shows the predefined format and legal range of fields in the parameter class section:

Entry Name	Entry Keyword	Field Number	Data Type	Required/Optional
Max Instances	MaxInst	1	UINT	Required
Parameter Class Descriptor	Descriptor	1	WORD	Required
Configuration Assembly Instance	CfgAssembly	1	UINT	Required

The CIP Object Library provides more specific details about the parameter fields, including the allowed data types and lengths. In general, the EDS parameter fields provide:

- **Max Instances** - Identifies the total number of configuration parameters contained in the device associated with the EDS
- **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

```
$ Sample Electronic Data Sheet illustrating the Parameter Class Section

[File]
...
[Device]
...
[ParamClass]
    MaxInst = 3;
    Descriptor = 0x0E;
    CfgAssembly = 3;

[Params]
...
[Groups]
...
```

### 7-3.5.5. Parameters Section

The parameters section identifies all of the configuration parameters in a device. All configuration performed on a device depends on the configuration parameters.

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. A white space or nothing between commas shall be used for optional fields not provided. Each entry contains the formatted fields shown in the table below. The “**ProxyParam**” and “**ProxiedParam**” keywords are defined further in 7-3.6, Modular EDS File Requirements.

Field Name	Field Number	Data Type	Required/Optional
Reserved	1	USINT	Required
Link Path Size	2	USINT	Required
Link Path	3	ASCII Character Array	Required
Descriptor	4	WORD	Required
Data Type	5	EPATH	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	<i>data type</i>	Required
Maximum Value	11	<i>data type</i>	Required
Default Value	12	<i>data type</i>	Required
Scaling Multiplier	13	UINT	Optional
Scaling Divider	14	UINT	Optional
Scaling Base	15	UINT	Optional

Field Name	Field Number	Data Type	Required/Optional
Scaling Offset	16	INT	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

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 provide:

- **Reserved** - This first field shall contain a zero.
- **Link Path Size** - The number of bytes used to represent path.
- **Link Path** - The link path identifier. The path shall be entered as a character array, using the path notation described in Section 4-3.4.8. If the attribute described by this **ParamN** entry is not directly addressable from the network, this field shall be empty. If 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.
- **Descriptor** - The parameter descriptor.
- **Data Type** - The data type identifier.
- **Data Size** -The numeric data size value in bytes.
- **Parameter Name** - The textual parameter name. If necessary, truncation of the retrieved text occurs to meet the maximum character array length allowed.
- **Units String** - The textual display units character array. If necessary, truncation of the retrieved text occurs to meet the maximum character array length allowed.
- **Help String** - The textual help character array. If necessary, truncation of the retrieved text occurs to meet the maximum character array length allowed.
- **Minimum Value** - The minimum numeric value that may be assigned to the parameter data value.
- **Maximum Value** - The maximum numeric value that may be assigned to the parameter data value.
- **Default Value** - The default numeric value assigned to the parameter data value.
- **Scaling Multiplier** - The numeric multiplier value applied to the current parameter data value.
- **Scaling Divider** - The numeric divisor value applied to the current parameter data value.
- **Scaling Base** - The numeric base value applied to the current parameter data value.
- **Scaling Offset** - The numeric offset value applied to the current parameter data value.
- **Multiplier Link** - The Parameter Object instance that contains the numeric multiplier value to apply to the current parameter data value.
- **Divisor Link** - The Parameter Object instance that contains the numeric divisor value to apply to the current parameter data value.
- **Base Link** - The Parameter Object instance that contains the numeric base value to apply to the current parameter data value.
- **Offset Link** - The Parameter Object instance that contains the numeric offset value to apply to the current parameter data value.
- **Decimal Precision** - The numeric precision value applied to the current parameter data value.

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.

### **7-3.5.6. Parameter Groups Section**

The parameter groups section 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. A comma separates all fields, and a semicolon indicates the end of the entry. The parameter group section contains:

Field Name	Field Number	Data Type	Required/Optional
Group Name String	1	ASCII Character Array	Required
Number of Members	2	UINT	Required
Parameter	3 thru (number of members + 2)	UINT	Required

```
$ Sample Electronic Data Sheet illustrating the Parameter Groups Section

[File]
...
[Device]
...
[ParamClass]
...
[Params]
...

[Groups]
    Group1 = "Setup", 2, 1, 2;                      $ group 1
    Group2 = "Monitor", 2, 2, 3;                     $ group 2
    Group3 = "Maintenance", 2, 1, 3;                 $ group 3
```

### 7-3.5.7. Assembly Section

The Assembly section 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.

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.

The entry keyword for all assemblies shall consist of one of the following character arrays, “Assem”, “ProxyAssem”, “ProxiedAssem” combined with the Assembly object instance number (decimal) for the device, e.g. “Assem1”. 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 “ProxyAssem” and “ProxiedAssem” keywords are defined further in 7-3.6, Modular EDS File Requirements.

Each entry shall contain the formatted fields shown in Table 7-3.1.

**Table 7-3.1. AssemN Keyword format**

Field name	Field number	Data type	Required/ Optional
Name	1	Eds_Char_Array	Optional
Path	2	Eds_Char_Array	Optional
Size	3	UINT	Conditional
Descriptor	4	WORD	Optional
Reserved	5, 6	empty	
Member Size	7, 9, 11 ...	UINT	Conditional
Member Reference	8, 10, 12 ...	AssemN, ParamN, Number, or EPATH	Conditional

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.

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 block described by this **AssemN** entry is not directly addressable from the link, this field shall be empty. If 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.

The third field, called "Size", shall be the size of the data block in bytes. Either this field or the "Member Size"/"Member Reference" fields shall be present. Both of these fields may be present; however, since they both specify the size of the buffer, the sizes specified by both means shall agree.

The fourth field, "Descriptor", shall be a bit-field that describes certain properties of the Assembly. The bits of this field can be interpreted as follows:

Bit	Name	Meaning
0	Allow Value Edit	If this bit is set (1), the contents of the Assembly's value member references can be edited. If reset (0), the contents of these member references may not be edited. The member references considered to be values are a Number and a Data Segment EPATH. If this field is empty, the meaning shall default to reset (0).
1-15		Reserved

Fields 5 and 6 shall be reserved for future definition and shall be empty.

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. 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;
- a string representing a path (EPATH);
- a number, maximum value  $2^{32}$ ;
- 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. The total of all member sizes shall equal the AssemN Size field (when expressed as bits).

For example, **Assem5** in Figure 7-3.2 shall be 1 byte long and have a default value of 0x21.

**Figure 7-3.2. [Assembly] section example**

```
[Params]
Param1 =
0,                                $ first field shall equal 0
6, "20 0F 24 01 30 01",           $ path size, path
0x0000,                            $ descriptor
2,                                 $ 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, 2, 2,                      "Fault state", "", "User Manual p49",
"Fault state", "", "User Manual p49",
0, 2, 2, 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, ;
```

### 7-3.5.8. Complete Electronic Data Sheet Example

A complete Electronic Data Sheet example is given below which is a compilation of the section examples given throughout this chapter.

```
$ Complete Electronic Data Sheet Example

[File]
DescText = "Smart Widget EDS File";
CreateDate = 04-03-94;                                $ created
CreateTime = 17:51:44;
ModDate = 04-06-94;                                    $ 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", 0x0E94, 1, 1,"Preset","V","User Manual p33",
0, 5, 1, 1, 1, 0, 0, 0, 0, 0, 2;

Param2 =                                              $ parameter instance
0,                                              $ First field shall equal 0
1, "20 04",                                     $ path size, path
0xA94,                                         $ descriptor - in hex format
1,                                              $ data type
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
```

### 7-3.5.9 Connection Manager Section

The Connection Manager section 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 7-3.6, Modular EDS File Requirements.

Each entry shall contain the formatted fields shown in Table 7-3.3.

**Table 7-3.3. Connection Manager section format**

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	Conditional
T⇒O RPI	6	UDINT or Param	Optional
T⇒O size	7	UINT or Param	Conditional
T⇒O format	8	Param or Assem	Conditional
config #1 size	9	UINT or Param	Optional
config #1 format	10	Param or Assem	Optional
config #2 size	11	UINT or Param	Optional
config #2 format	12	Param or Assem	Optional
Connection name string	13	Eds_Char_Array	Required
Help string	14	Eds_Char_Array	Required
Path	15	Eds_Char_Array	Required

#### Trigger and transport mask

The bit assignments for the trigger and transport mask shall be as shown in Table 7-3.4. 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 7-3.4. Trigger and transport mask bit assignments**

Bit	Bit definition
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

**Connection parameters**

The bit assignments for the connection type and priority mask shall be as shown in Table 7-3.5. 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 7-3.5. Connection parameters bit assignments**

Bit	Bit definition
0	O⇒T fixed size supported
1	O⇒T variable size supported
2	T⇒O fixed size supported
3	T⇒O variable size supported
4 – 5	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	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

Bit	Bit definition
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

## 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.

## O⇒T size

The O⇒T size shall be the number of bytes delivered to the target transport. It shall not include the transport sequence count. The O⇒T size shall be a UINT or a Param entry from the [Params] section that evaluates to a UINT. If this field is empty, the defined size of the O⇒T format shall be used after the optional run/idle header size is added.

## 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 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 $\Rightarrow$ T size field is empty. The O $\Rightarrow$ T format shall not include the 32-bit run/idle header if it is present.

### **T $\Rightarrow$ O RPI**

The T $\Rightarrow$ O RPI shall be the number of microseconds of the requested packet interval. The T $\Rightarrow$ 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 $\Rightarrow$ O RPI.

### **T $\Rightarrow$ O size**

The T $\Rightarrow$ O size shall be the number of bytes produced by the target transport. It shall not include the transport sequence count. The T $\Rightarrow$ O size shall be a UINT or a Param entry from the [Params] section that evaluates to a UINT. If this field is empty, the defined size of the T $\Rightarrow$ O format shall be used after the optional run/idle header is added.

### **T $\Rightarrow$ O format**

The T $\Rightarrow$ 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 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 $\Rightarrow$ O size field is empty. The format shall include the status header if it is present.

### **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 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.

### **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.

### 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.

### Path

A path referencing the target object. The path shall be entered as a CIP Path (character array). In addition, the path field may also contain the other following references:

Param entries from the [Params] section;

- the keyword SLOT;
- the keyword SYMBOL\_ANSI;
- the keyword SLOT\_MINUS\_ONE.

The Param entries 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 . 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. 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. The keyword, SYMBOL\_ANSI, 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.

**Figure 7-3.6. [Connection Manager] section example**

```
[Params]
Param1 =
    0, , ,
    0x0004,
    8, 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,
    8, 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 & realtime format
                           $ fixed, 32-bit headers, scheduled,
                           $ O=>T point-to-point, T=>O multicast
    , 16, ,
    , 12, ,
    , ,
    , ,
    "read/write",         $ connection name
    "",                  $ Help string
    "20 04 24 01 2C [Param2] 2C [Param1]";
```

## Port Section

The Port section 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.

Field Name	Field number	Data type	Required/Optional
Port Type Name	1	Field Keyword	Required
Port Name	2	Eds_Char_Array	Optional
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

The first field, called “Port Type Name”, shall be one of

- ControlNet
- ControlNet\_Redundant
- TCP<sup>1</sup>
- DeviceNet

A vendor-specific value beginning with the device’s Vendor ID and an underscore character ('65535\_')

<sup>1</sup> This shall indicate an EtherNet/IP capable TCP port.

```
[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 optional “Name” field shall be a string giving a name to the port, and may be used by a user interface. 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 can not send CIP messages shall not have a port number 1.

### 7-3.6. Modular EDS File Requirements

This section describes the concept and contents of a Modular EDS and specifies the usage requirements.

### 7-3.6.1. Modular Section

The [Modular] section 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 7-3.7. 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 7-3.7. [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
    0x0004,                 $ descriptor
    8,                      $ data type: 32-bit Unsigned Long 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 a single field that indicates how many slots of the chassis are consumed by the module. 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 7-3.8.

**Table 7-3.8. 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 7-3.9.

**Figure 7-3.9. [Modular] section describing a module with a network connection**

```
[Modular]
Width = 1;

Rack1 =
    65535, 101, 1, 1, 1, ,,,,
    0;
$ this module can only plug into
$ slot 0 of this five slot chassis

Rack2 = 65535, 101, 2, 1, 1, ,,,,
0;
```

## 7-3.6.2. Modular Additions to Basic EDS Sections

### 7-3.6.2.1. 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 “**ExternIDExactMatch**”. 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 “**ExternIDExactMatch**” 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 address to a module in slot 0. The fourth field shall be the ExternalID returned when an empty slot exists.

**Figure 7-3.10. [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";

ExternIDExactMatch = No;
```

### 7-3.6.2.2. Additions to the Parameter Section

The “**ProxyParam**” and “**ProxiedParam**” keywords shall be used to describe parameters that are proxied by a ControlNet 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**”.

### 7-3.6.2.3. 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**” and “**ProxiedAssem**” 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**” keyword shall exist in the EDS for the device that performs the proxy; the “**ProxiedAssem**” 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 “**ProxyAssem**” keywords to EDS files containing “**ProxiedAssem**” keywords. This association shall exist when both EDS files specify a matching Rack entry.

The decimal number (that is combined with “**ProxyAssem**” and “**ProxiedAssem**”) shall be used to match a “**ProxyAssem**” to a “**ProxiedAssem**”. The field values of a matched “**ProxyAssem**” and “**ProxiedAssem**” pair shall be combined to constitute the same field value information that exists in a single “**Assem**” entry. This combination shall be done by using the field value from the “**ProxyAssem**” unless that field value is one of the keywords “**Module**” or “**ModuleMemberList**”. When the field value specified in the “**ProxyAssem**” is “**Module**” the field value specified in the “**ProxiedAssem**” 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**” is “**ModuleMemberList**” all “Member Size” and “Member Reference” fields specified in the “**ProxiedAssem**” shall be used. It shall be legal to specify field values for “**ProxiedAssem**” entries whose corresponding field value in the “**ProxyAssem**” is not “**Module**”, however, these field values shall not be used, they shall exist only for documentation.

**Figure 7-3.11. Example of ProxyParam and ProxyAssem**

```
[Params]
Param1 = 0,,,"0x0010,2,2," Target Error Codes","","","",0,0xFFFF,0,0,0,0,0,0,0,0,0;
ProxyParam1 = 0,,,"0x0000,2,2,"input size","","","",Module,Module,Module,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 7-3.12. Example of ProxiedParam and ProxiedAssem**

```
[Params]
ProxiedParam1 = ,,,,,"input size","","","",0,2,2,,,,;;
[Assembly]
ProxiedAssem1 = "real time input format",,,;;
ProxiedAssem2 = "real time status format",,,,,16,;

[Connection Manager]
ProxiedConnect1 = ,,,0,,,,,,;;
```

#### 7-3.6.2.4. 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 “**ProxiedConnect**” 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 intentionally left blank

## **Volume 1: CIP Common Specification**

### **Chapter 8: Physical Layer**

This page is intentionally left blank

## 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 Volume 2 (of the appropriate CIP network adaptation) for physical layer behavior defined on that network.

Examples of CIP Networks include: DeviceNet, ControlNet, and EtherNet/IP.

This page is intentionally left blank

**Volume 1: CIP Common Specification**

**Chapter 9: Indicators and Middle Layers**

This page is intentionally left blank

## 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 Volume 2 (of the appropriate CIP network adaptation) for indicator or middle layer behavior defined on that network.

Examples of CIP Networks include: DeviceNet, ControlNet, and EtherNet/IP.

This page is intentionally left blank

## **Volume 1: CIP Common Specification**

### **Chapter 10: Bridging and Routing**

---

This page is intentionally left blank

## 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 consistant I/O transaction times.

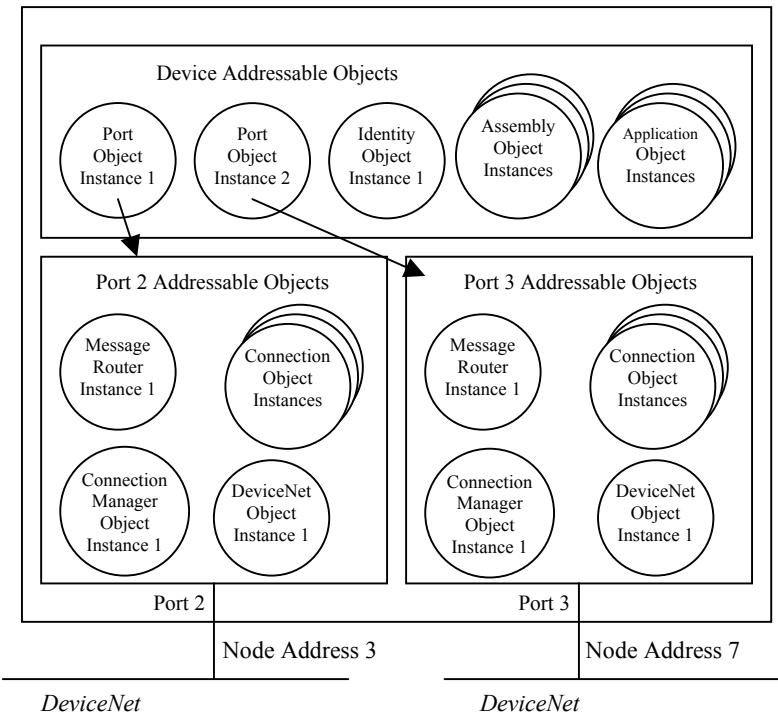
Each Port Segment represents a ‘hop’ in the path, from one subnet to another, and the CIP router removes it’s Port Segment from the path before forwarding the service to the next destination. The Port Segment tells the router which port (subnet) to send the message on and the node address of the destination 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 an Offlink 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 <sup>2</sup>
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.
<p>Failure to execute the Explicit Messaging transaction with the Remote Target Device. This is due to:</p> <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>	<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 value zero (0).</li> </ul>
<p>Failure to execute the Unconnected Send transaction with the next CIP Router in the path. This is due to:</p> <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>	<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>
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:
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).
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

Event Description	Action To Take <sup>2</sup>
	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 an 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 10-4.1 and 10-4.2 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 a 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.

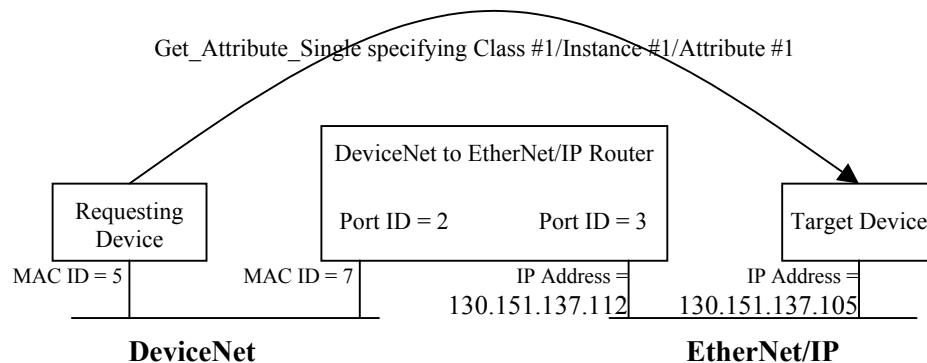
## 10-6 EXAMPLES

### 10-6.1 Using the Unconnected Send Service For Explicit Messaging – Single Hop

This section presents examples to further clarify the operation of the Connection Manager Object and the Unconnected Send service.

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**

<b>Bytes</b>	<b>Meaning</b>
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**

<b>Bytes</b>	<b>Meaning</b>
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 0F 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 dependant 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:

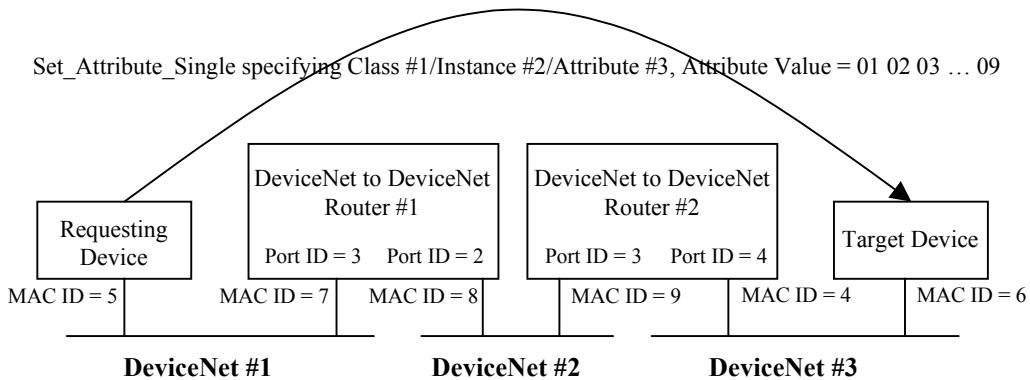
**Table 10-6.6 Routing Failure Explicit Message Response on DeviceNet**

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 <i>Remaining PathSize</i> 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 <i>Transaction_Id</i> field inserted by the router.
07	<i>Priority/Tick Time</i> field (Time Tick = 128 ms).
08	<i>Connection Timeout Ticks</i> field (8 Ticks which results in 1024 ms timeout, 512 ms less than was originally received).
11 00	<i>Message_Size</i> field = 17 bytes.
10	<i>Service</i> field = <i>Set_Attribute_Single</i> .
03	<i>Request_Path_Size</i> field indicates that there are 3 words in the request path.
20 01 24 02 30 03	<i>Request_Path</i> 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	<i>Request_Data</i> field specifying the attribute data associated with the <i>Set_Attribute_Single</i> request.
00	Pad byte.
01	<i>Route_Size</i> field indicates that there is 1 word in the <i>Route_Path</i> .
00	Reserved byte.
04 06	<i>Route_Path</i> 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.

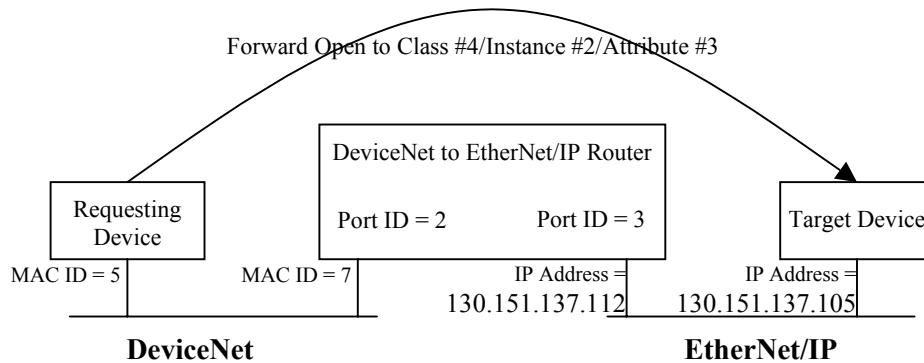
- Since there is no additional path routing, the device at MAC ID 6 is the Remote Target Device. The DeviceNet Router establishes an Explicit Messaging Connection with that device (if necessary), extracts the actual transaction from the Message Request portion of the Service Data, and delivers the Explicit 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	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

Byte Value	Meaning
31 33 37 2E 31 30 35 21 00 04 00 25 00 02 00 30 03 00	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 values 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 an 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 by.
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 an 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.

## **Volume 1: CIP Common Specification**

### **Appendix A: Explicit Messaging Services**

---

This page is intentionally left blank

## 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)

Service Code (in hex)	Service Name
15	Restore
16	Save
17	No Operation (NOP)
18	Get Member
19	Set Member
1A	Insert Member
1B	Remove Member
1C-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.

**Get\_Attribute\_All**

Returns the contents of all attributes of an object or class.

Service Code: 01<sub>hex</sub>

**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

Refer to Chapter 4 within of the CIP Common Specification for a more detailed discussion of this service.

**Request Service Data Field Parameters**

NONE

**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.

**Set\_Attributes\_All**

Modifies the contents of the attributes of the class or object.

Service Code: 02<sub>hex</sub>

**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 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 Set\_Attributes\_All success response is returned.

Refer to Chapter 4 within the CIP Common Specification for a more detailed discussion of this service.

**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.

**Success Response Service Data Field Parameters**

NONE

**Get\_Attribute\_List**

The Get\_Attribute\_List service shall return the contents of the selected gettable attributes of the specified object class or instance.

Service Code: 03<sub>hex</sub>

**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.

**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

### 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-4.4. 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 H)
	Object/Class specific attribute data	“Attribute Data”, Attribute response data - exists when “Attribute Status” is the value “Success” (0x00).

**Set\_Attribute\_List**

The Set\_Attribute\_List service shall set the contents of selected attributes of the specified object class or instance.

Service Code: 04<sub>hex</sub>

**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.

**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-4.4. 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

### 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-4.5. 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 H)
	Object/Class specific attribute data	“Attribute Data”, Attribute response data - may exist when “Attribute Status” is the value “Success” (0x00).

**Reset**

Invokes the *Reset* service of the specified Class/Object. Typically this would cause a transition to a default state or mode.

Service Code: 05<sub>hex</sub>

**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.

**Request Service Data Field Parameters**

The following information is **OPTIONALLY** specified within the Service Data Field of a Reset request.

**Table A-4.6. 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.

**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-4.7. 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.

**Start**

Invokes the *Start* service of the specified Class/Object. Typically this would place an object into a running state/mode.

Service Code: 06<sub>hex</sub>

**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.

**Request Service Data Field Parameters**

The following information is **OPTIONALLY** specified within the Service Data Field of a Start request.

**Table A-4.8. 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.

**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-4.9. 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.

**Stop**

Invokes the *Stop* service of the specified Class/Object. Typically this would place an object into a stopped or idle state/mode.

Service Code: 07<sub>hex</sub>

**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.

**Request Service Data Field Parameters**

The following information is **OPTIONALLY** specified within the Service Data Field of a Stop request.

**Table A-4.10. 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.

**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-4.11. 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.

**Create**Service Code: 08<sub>hex</sub>

Results in the instantiation of a new object within the specified class.

**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.

**Request Service Data Field Parameters**

The following information is **OPTIONALLY** specified within the Service Data Field of a Create request.

**Table A-4.12. 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.

**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-4.13. 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.

**Delete**

Deletes an object instance of the specified class.

Service Code: 09<sub>hex</sub>

**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.

**Request Service Data Field Parameters**

The following information is **OPTIONALLY** specified within the Service Data Field of a Delete request.

**Table A-4.14. 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.

**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-4.15. 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.

**Multiple Service Packet**Service Code: 0A<sub>hex</sub>

Performs a set of services as an autonomous sequence.

**Service Requirements**

The following list details requirements associated with the Delete 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 results 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.

**Request Service Data Field Parameters**

The following information is specified within the Service Data Field of a Multiple Service Packet request.

**Table A-4.16. 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.

**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-4.17. 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.
	UINT	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.

**Apply\_Attributes**

Causes attribute values whose use is *pending* to become actively used.

Service Code: 0D<sub>hex</sub>

**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.

**Request Service Data Field Parameters**

The following information is OPTIONALLY specified within the Service Data Field of an Apply\_Attributes request.

**Table A-4.18. 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.

**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-4.19. 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.

**Get\_Attribute\_Single**

Returns the contents of the specified attribute.

Service Code: 0E<sub>hex</sub>

**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.

**Request Service Data Field Parameters**

The following information is specified within the Service Data Field of a Get\_Attribute\_Single request.

**Table A-4.20. 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.

**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-4.21. 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

**Set\_Attribute\_Single**

Modifies an attribute value.

Service Code: 10<sub>hex</sub>

**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.

**Request Service Data Field Parameters**

The following information is specified within the Service Data Field of a Set\_Attribute\_Single request.

**Table A-4.22. 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.

**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-4.23. 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.

**Find\_Next\_Object\_Instance**

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.

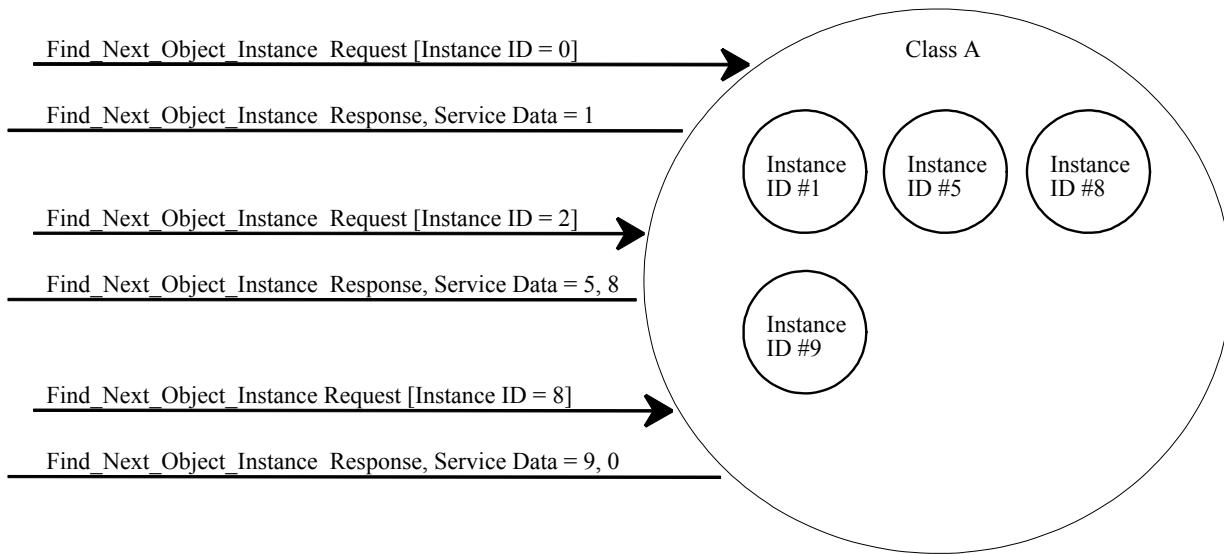
Service Code: 11<sub>hex</sub>

**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.



### Request Service Data Field Parameters

The following information is specified within the Service Data Field of a Find\_Next\_Object\_Instance request.

**Table A-4.24. 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.

### 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-4.25. 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.

**Restore**

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.

Service Code: 15<sub>hex</sub>

**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.

**Request Service Data Field Parameters**

The following information is **OPTIONALLY** specified within the Service Data Field of a Restore request.

**Table A-4.26. 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.

### 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-4.27. 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.

**Save**

Copies the contents of an class/object's attributes to a location accessible by the Restore service.

Service Code: 16<sub>hex</sub>

**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.

**Request Service Data Field Parameters**

The following information is **OPTIONALLY** specified within the Service Data Field of a Save request.

**Table A-4.28. 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.

**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-4.29. 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.

**No Operation (NOP)**

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.

Service Code: 17<sub>hex</sub>

**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.

**Request Service Data Field Parameters**

NONE

**Success Response Service Data Field Parameters**

NONE

**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.**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.

**Request Service Data Field Parameters**

Request Field	Required/Optional/Not Present
Member Data	Not present

**Success Response Service Data Field Parameters**

Response Field	Required/Optional/Not Present
Member ID	Required
Member Data	Required

**Set\_Member**

Sets member(s) information in an attribute.

Service Code: 19<sub>hex</sub>

See Section A-5 for *Member Service Protocol* details.

**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.

**Request Service Data Field Parameters**

Request Field	Required/Optional/Not Present
Member Data	Required

**Success Response Service Data Field Parameters**

Response Field	Required/Optional/Not Present
Member ID	Conditional (Required if Member Data is present)
Member Data	Optional

**Insert\_Member**Service Code: 1A<sub>hex</sub>

Inserts member(s) into an attribute.

See Section A-5 for *Member Service Protocol* details.**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.

**Request Service Data Field Parameters**

Request Field	Required/Optional/Not Present
Member Data	Optional

**Success Response Service Data Field Parameters**

Response Field	Required/Optional/Not Present
Member ID	Required
Member Data	Optional

**Remove\_Member**

Removes member(s) from an attribute.

Service Code: 1B<sub>hex</sub>

See Section A-5 for *Member Service Protocol* details.

**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.

**Request Service Data Field Parameters**

Request Field	Required/Optional/Not Present
Member Data	Not present

**Success Response Service Data Field Parameters**

Response Field	Required/Optional/Not Present
Member ID	Required
Member Data	Required

## 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
2-63hex	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-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**

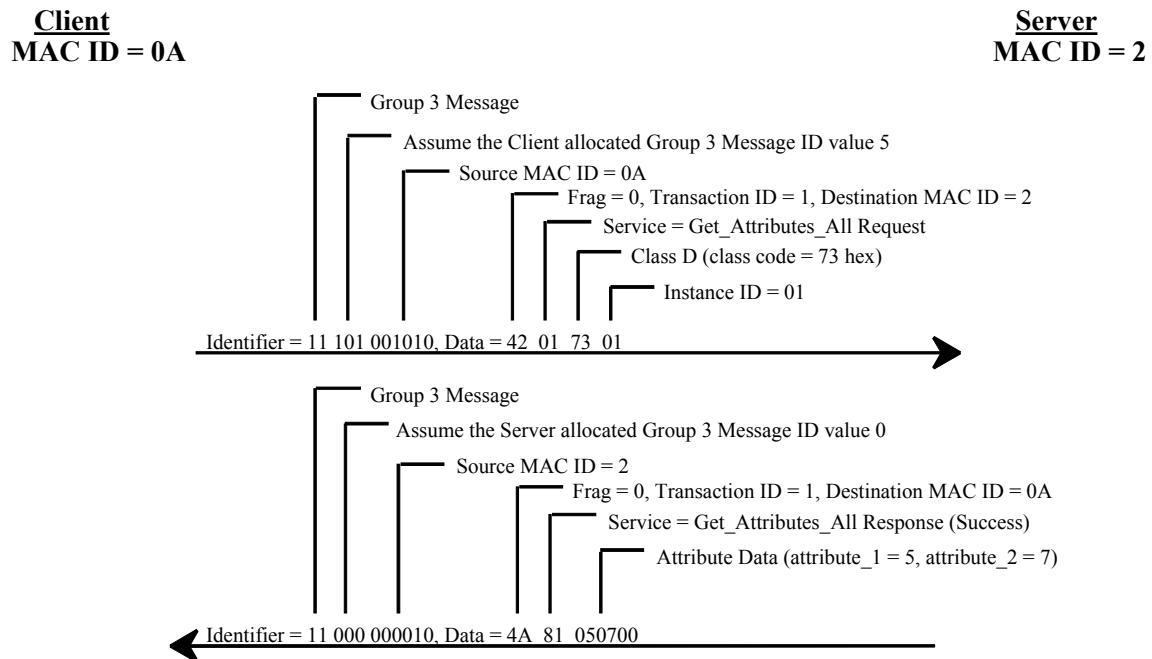
Attribute Name	Attribute ID	Format	Access Method	Attribute Description
attribute_1	1	USINT	Gettable/ Settable	No limit to value
attribute_2	2	UINT	Gettable/ Settable	No limit to value

## 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 a Explicit Messaging Connection has already been established and, as such, the Connection IDs associated with the Explicit Messaging Connections have already been agreed upon.**

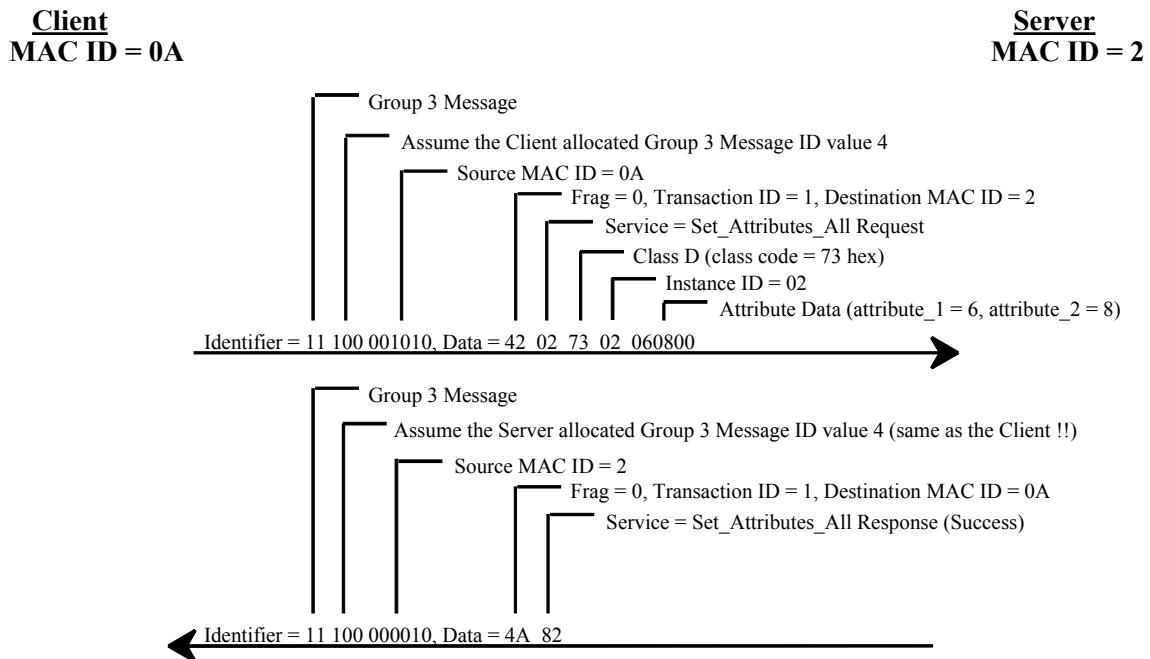
**Get\_Attributes\_All**

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).



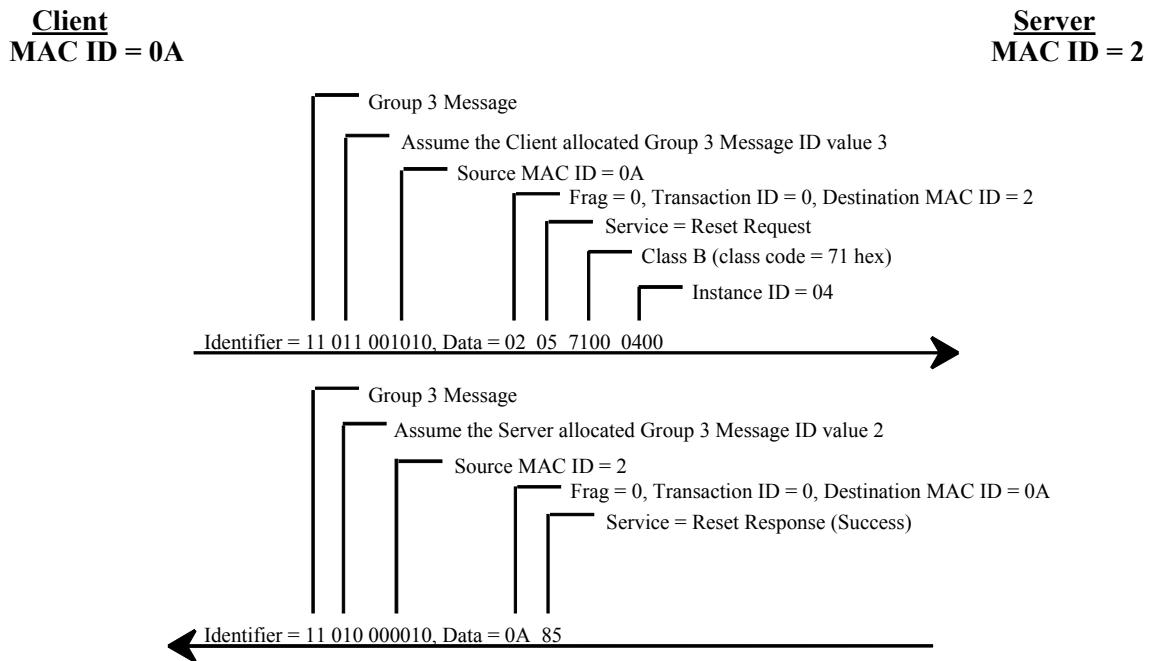
**Set\_Attributes\_All**

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).



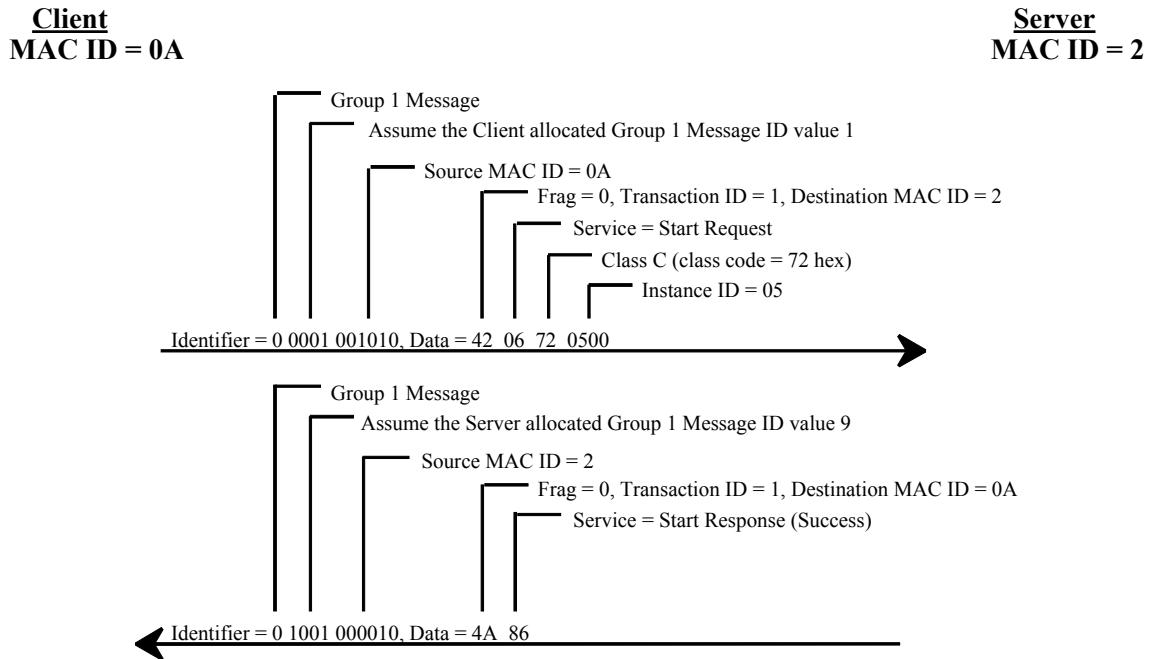
**Reset**

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).



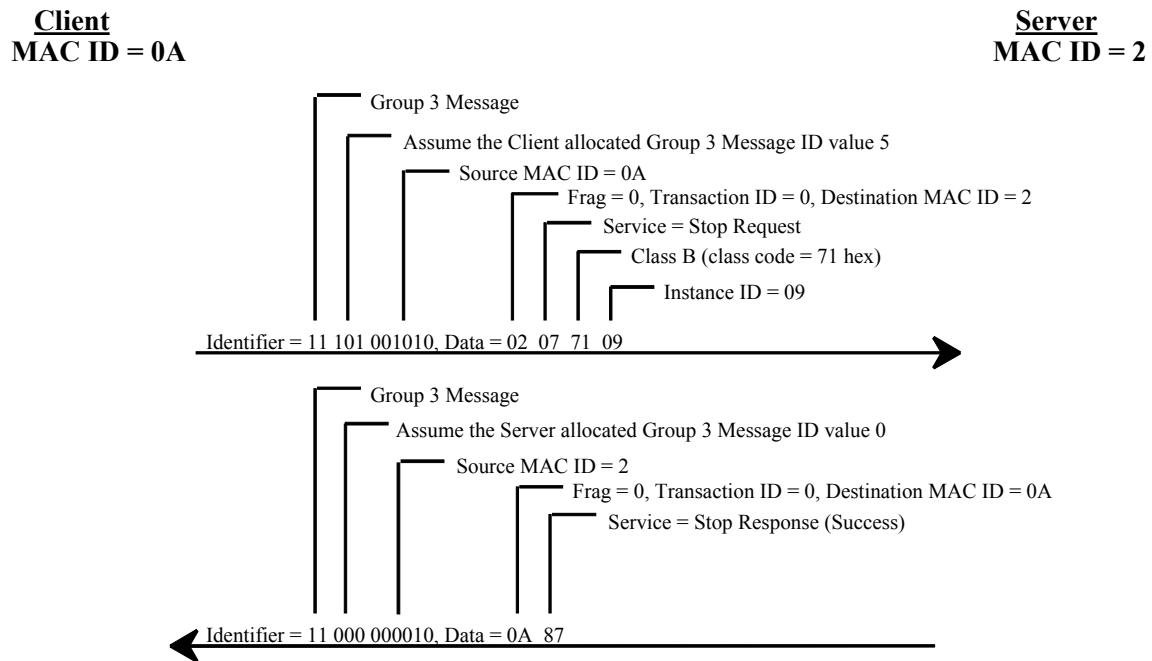
**Start**

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).



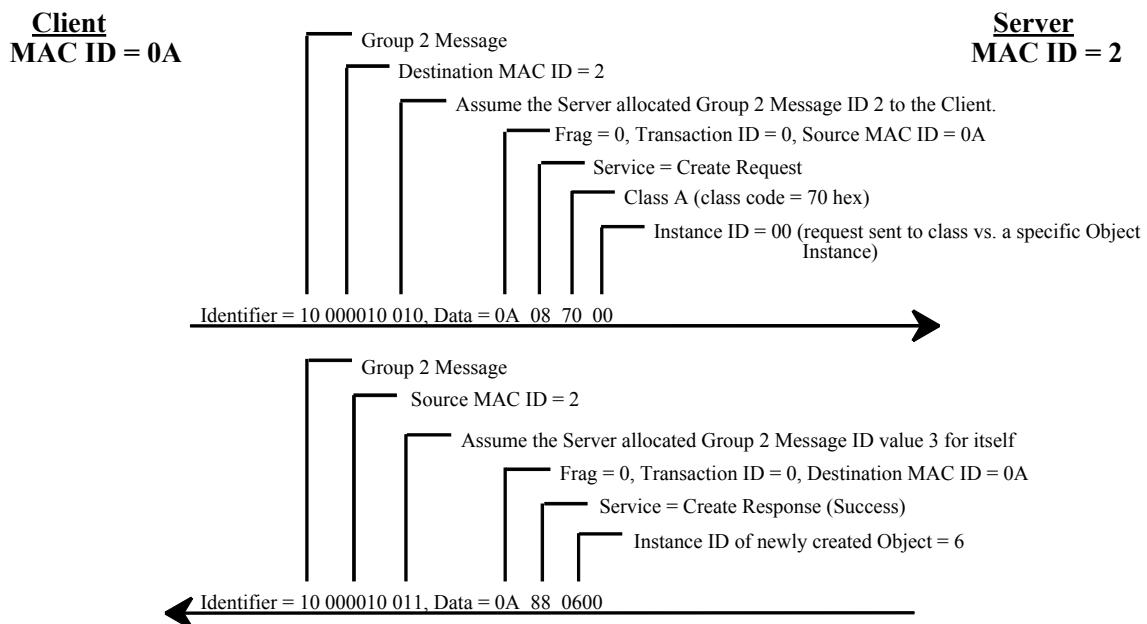
**Stop**

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).



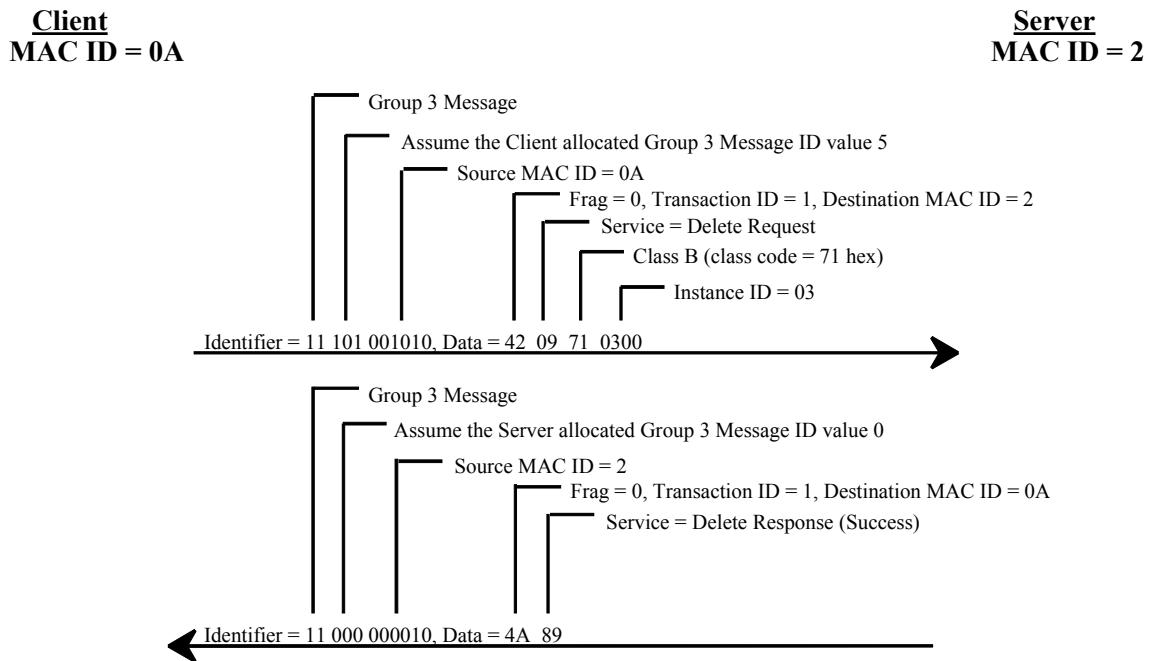
**Create**

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)



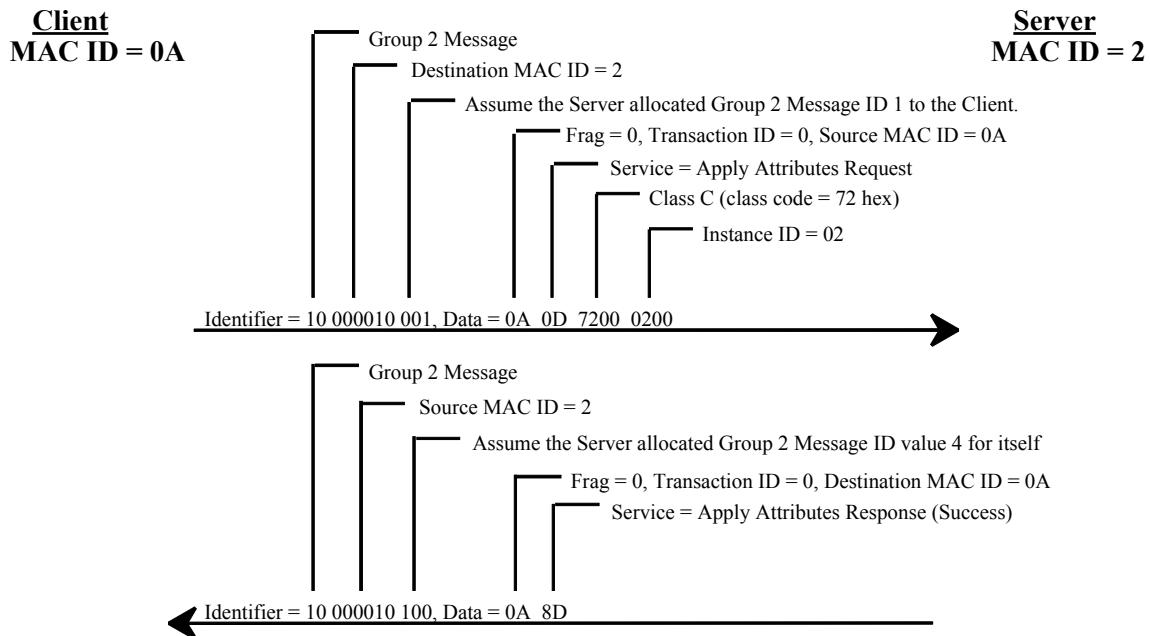
**Delete**

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).



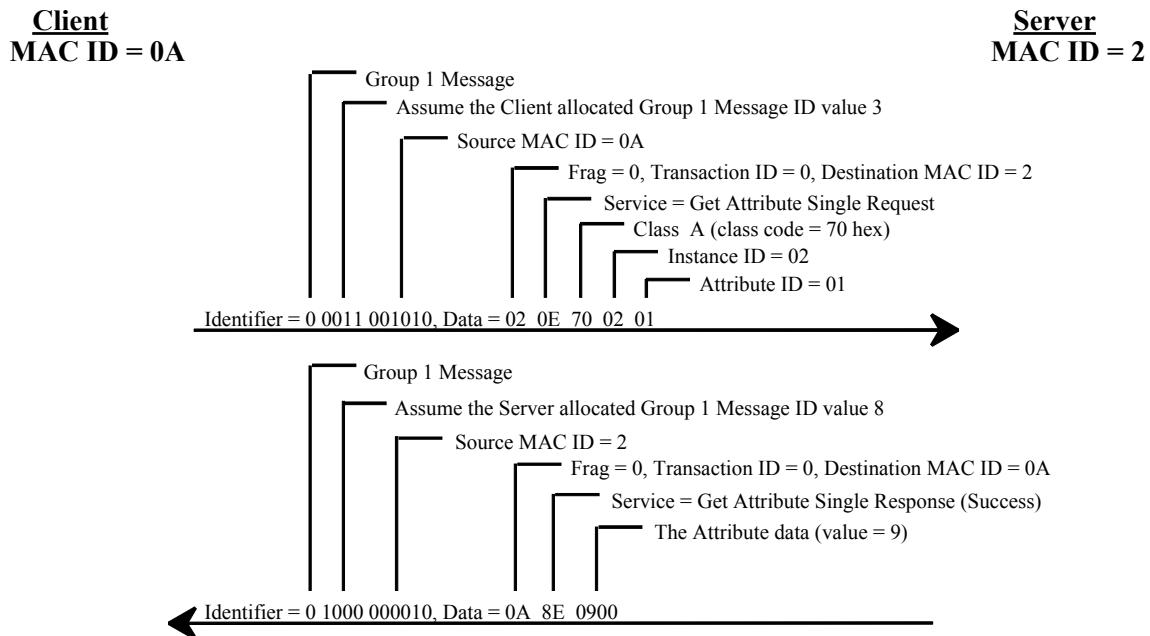
**Apply Attributes**

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).



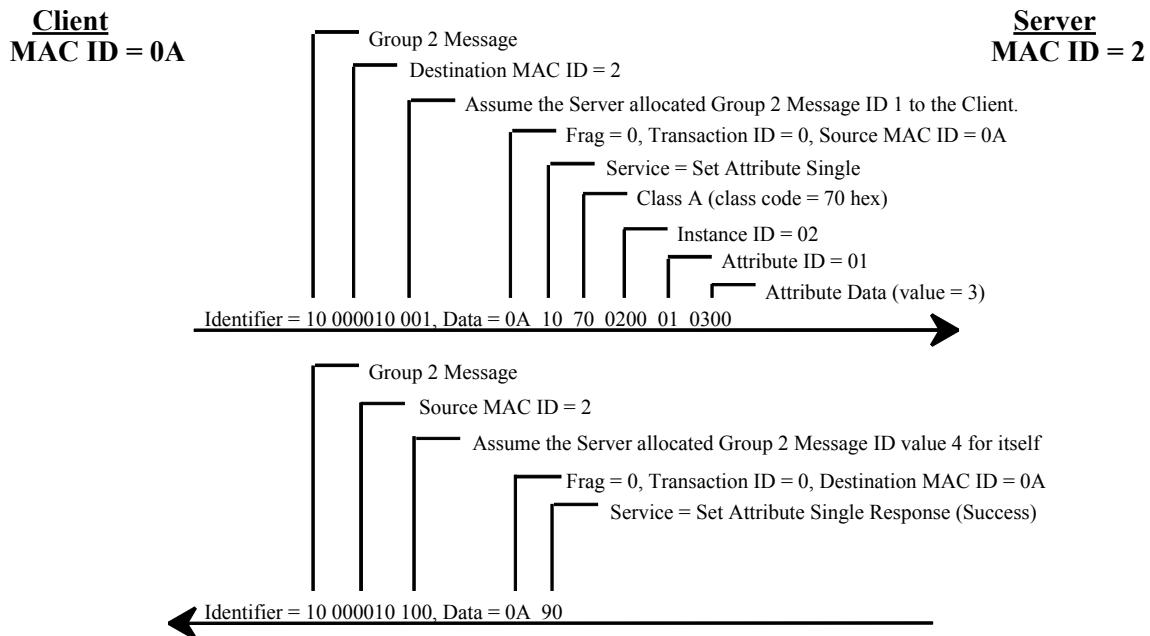
**Get\_Attribute\_Single**

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).



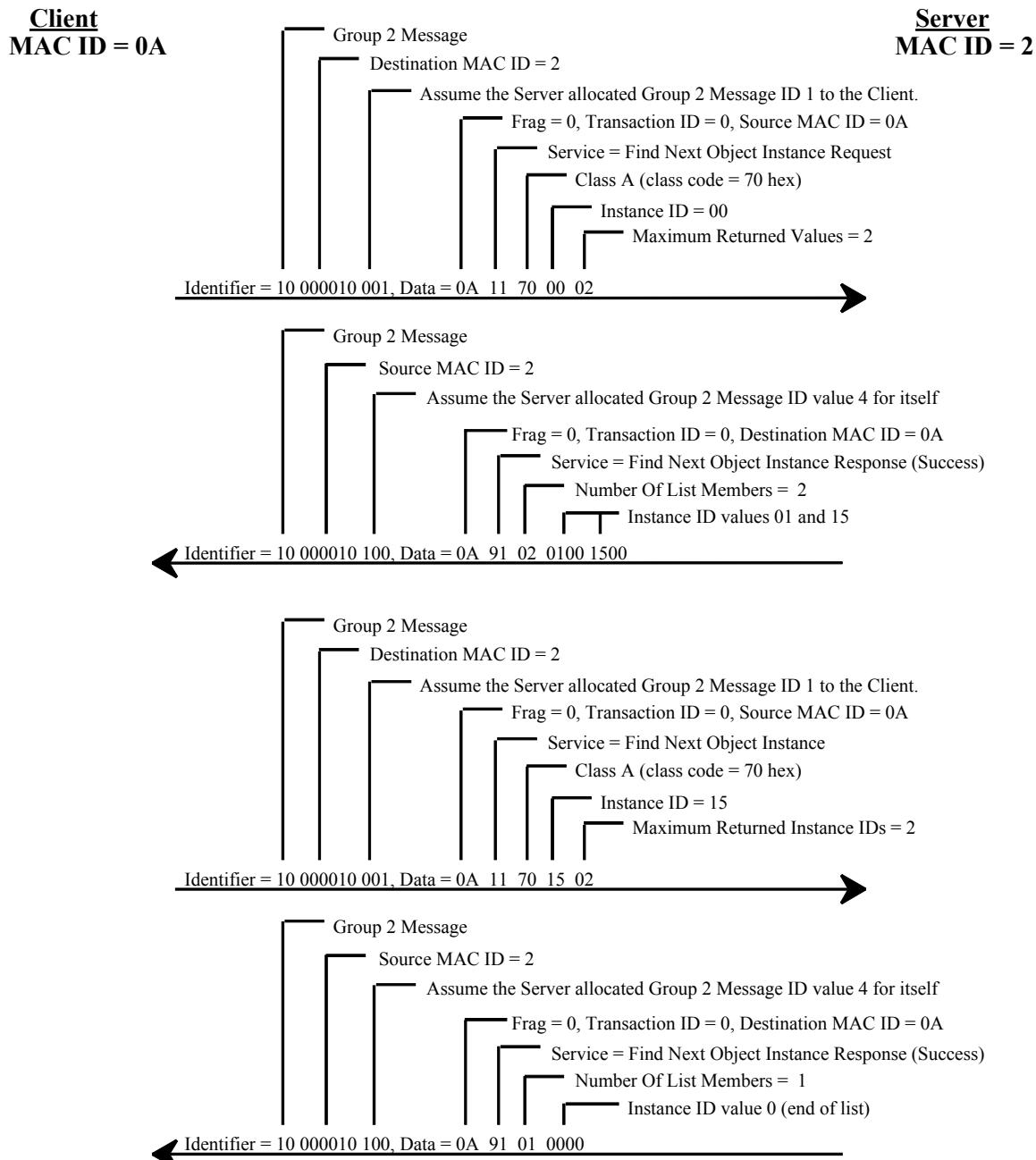
**Set\_Attribute\_Single**

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).



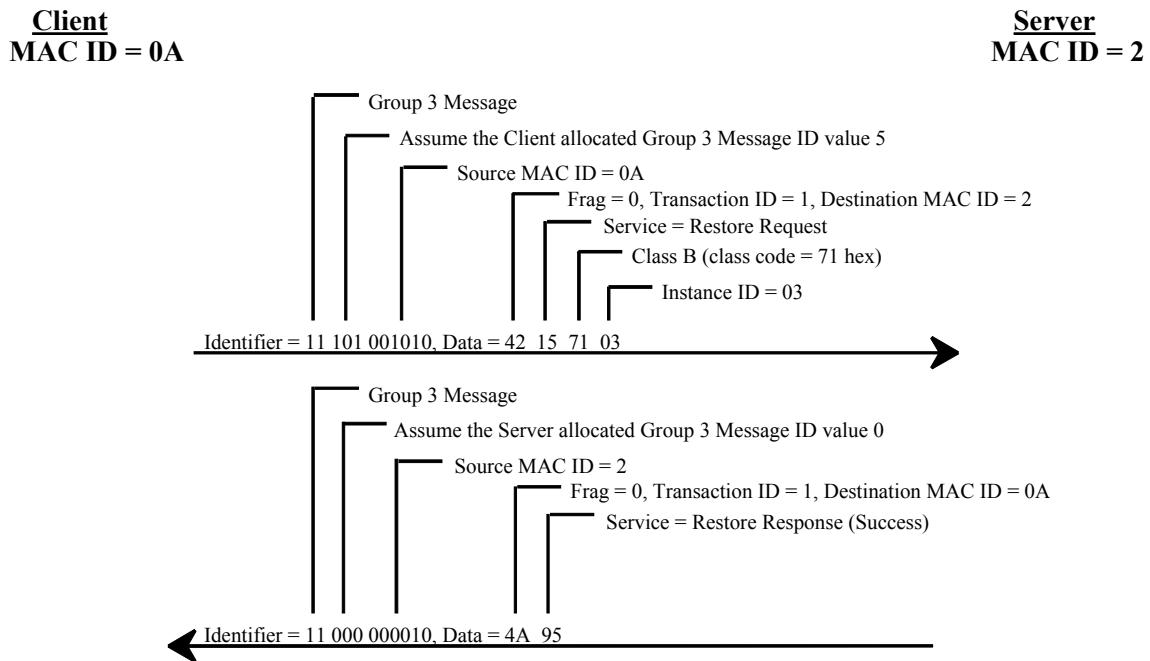
**Find\_Next\_Object\_Instance**

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.



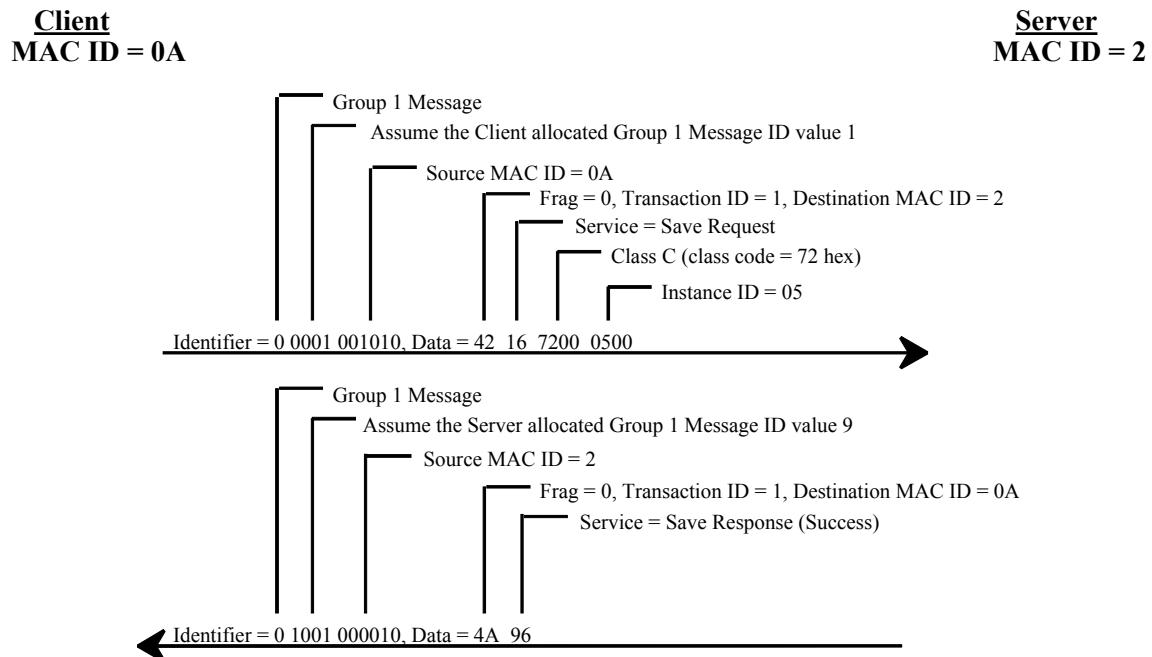
**Restore**

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).



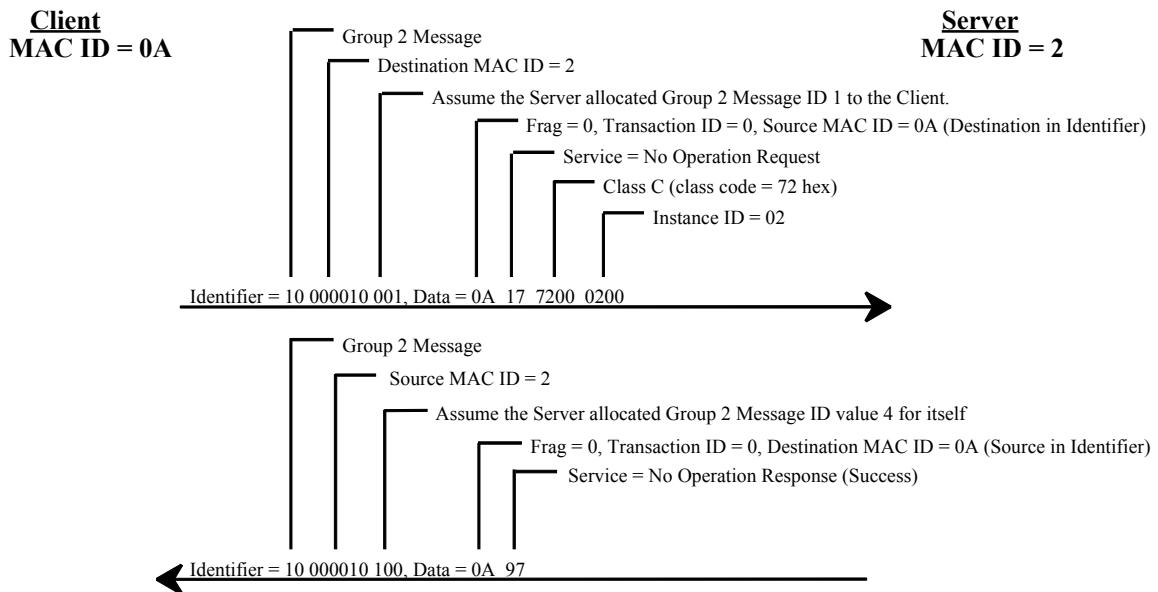
**Save**

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).



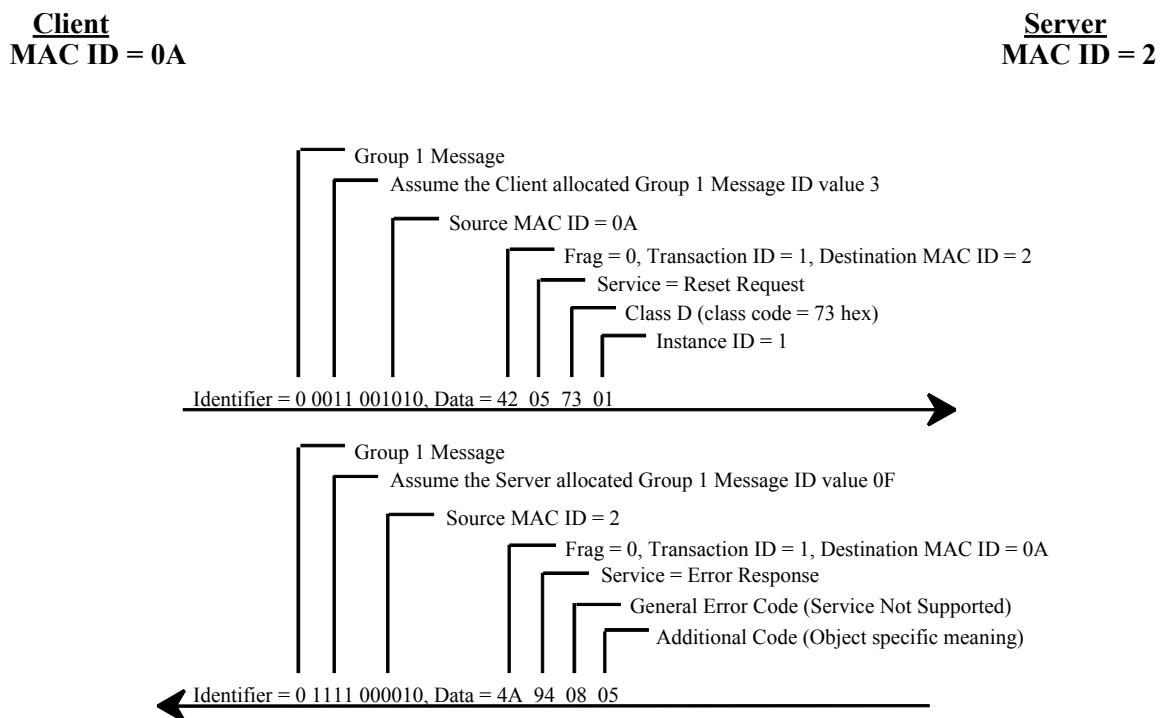
**No Operation**

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).



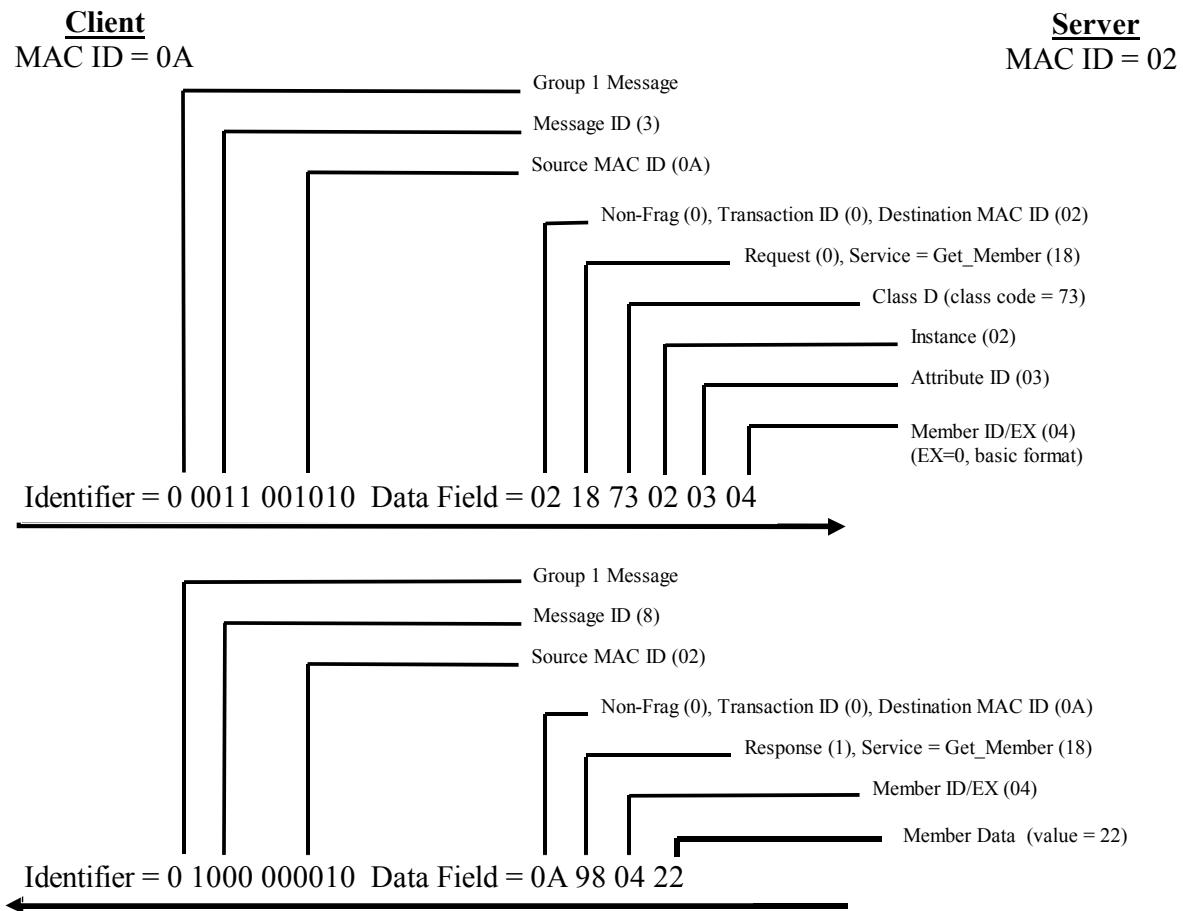
**Error Response**

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.



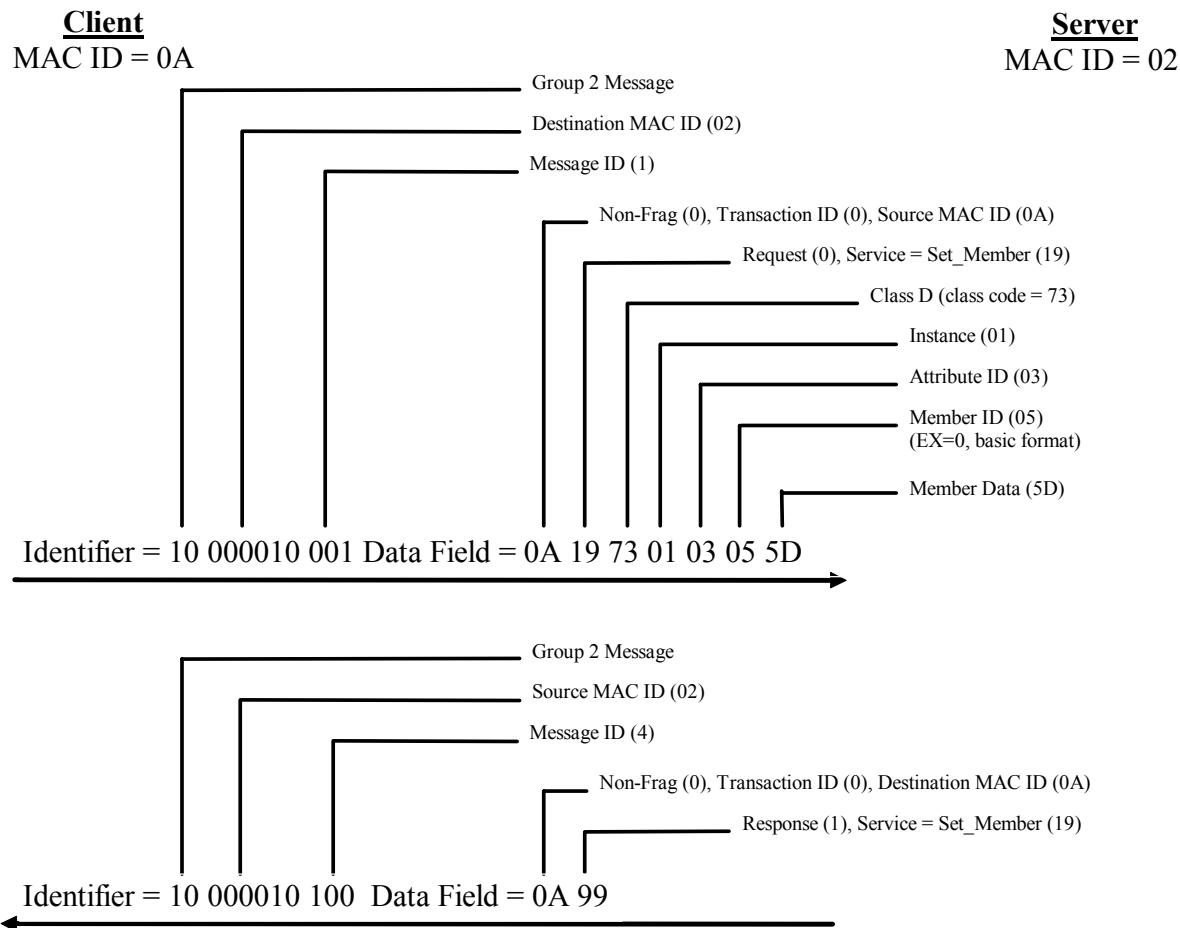
**Get\_Member**

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).



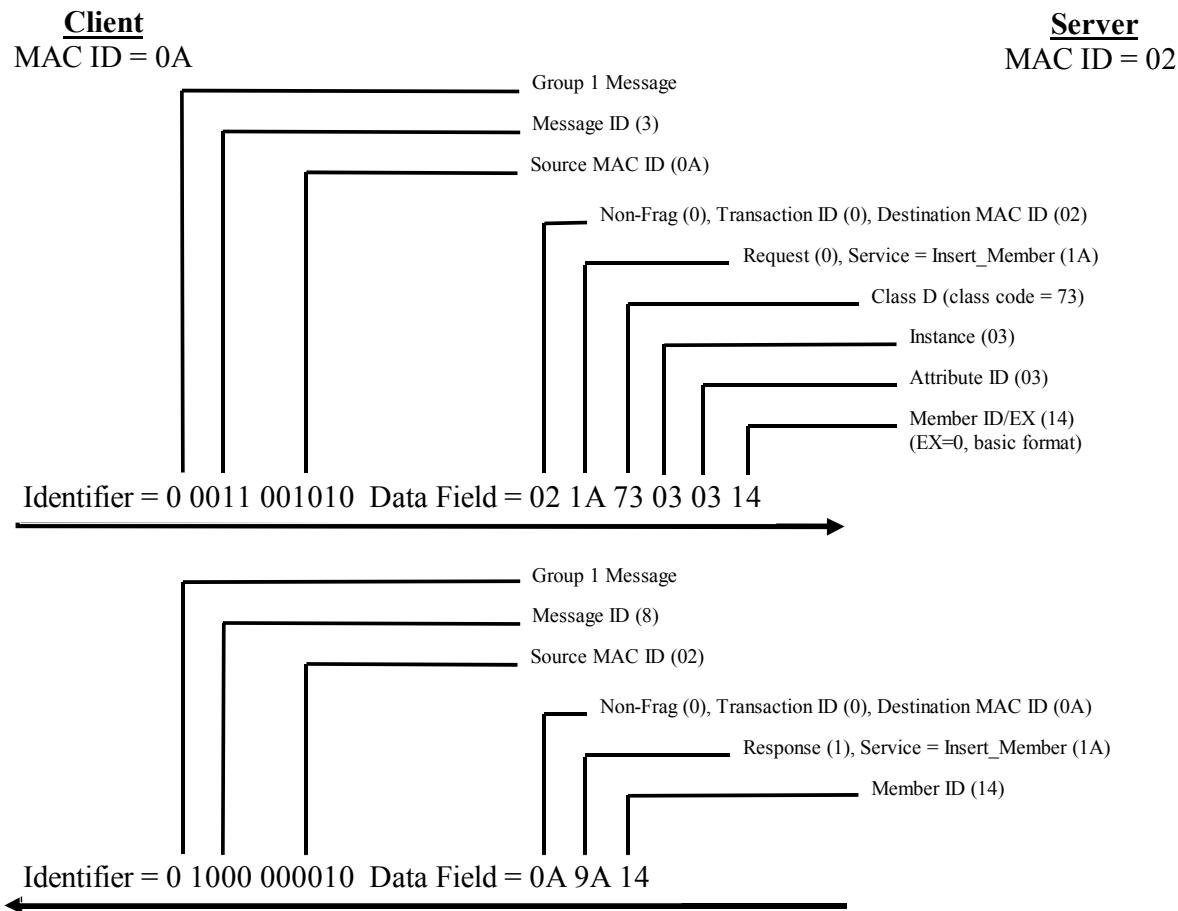
**Set\_Member**

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).



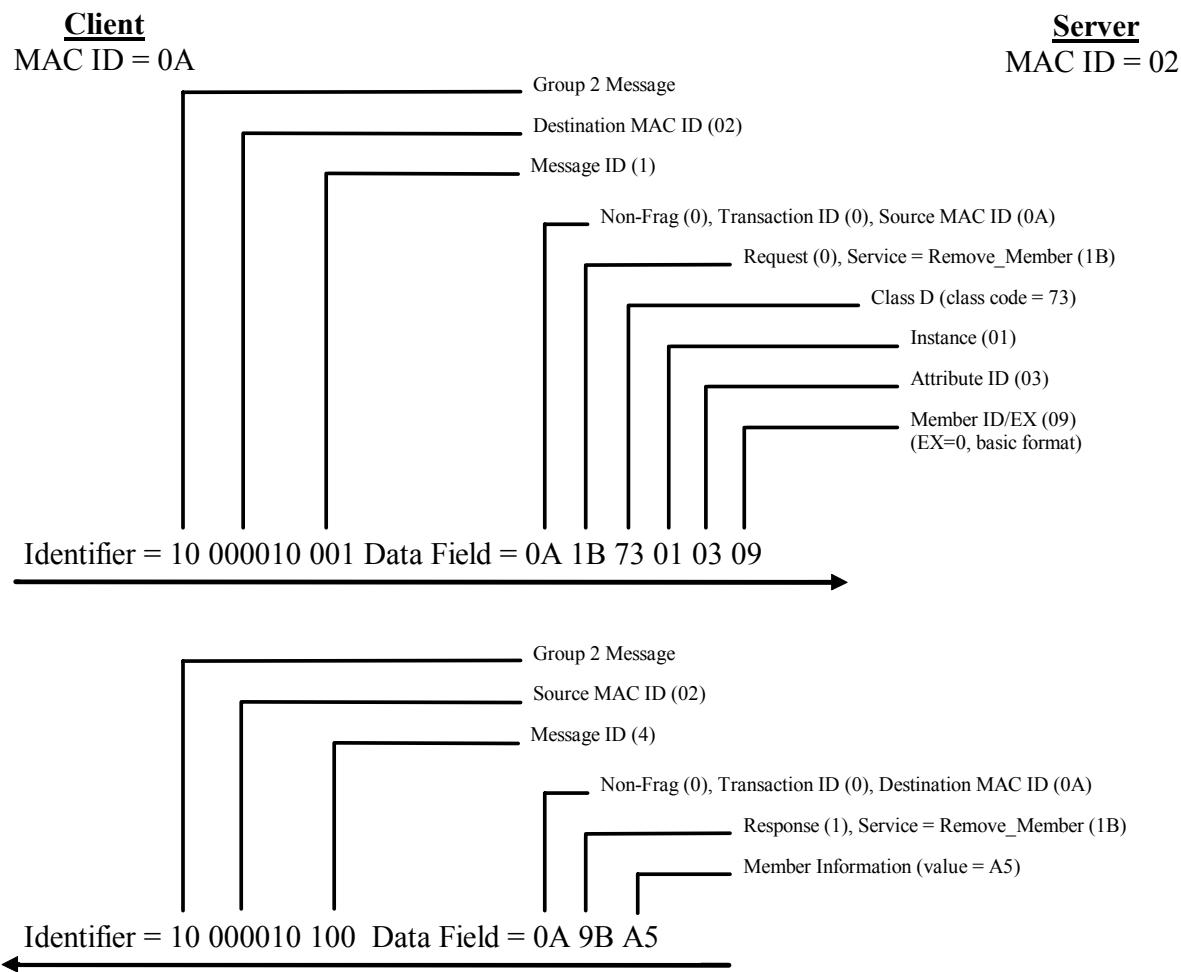
**Insert\_Member**

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).



**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).



This page is intentionally left blank

## **Volume 1: CIP Common Specification**

## **Appendix B: Status Codes**

---

This page is intentionally left blank

## 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.

<b>General Status Code (in hex)</b>	<b>Status Name</b>	<b>Description of Status</b>
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.
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 behaviour.
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.
23-24		Reserved by CIP for future extensions
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 <b>Service not supported</b> , <b>Attribute not supported</b> and <b>Attribute not settable</b> .
2B - 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: CIP Common Specification**

## **Appendix C: Data Management**

---

This page is intentionally left blank

## C-1 ABSTRACT SYNTAX ENCODING 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:

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
- 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.

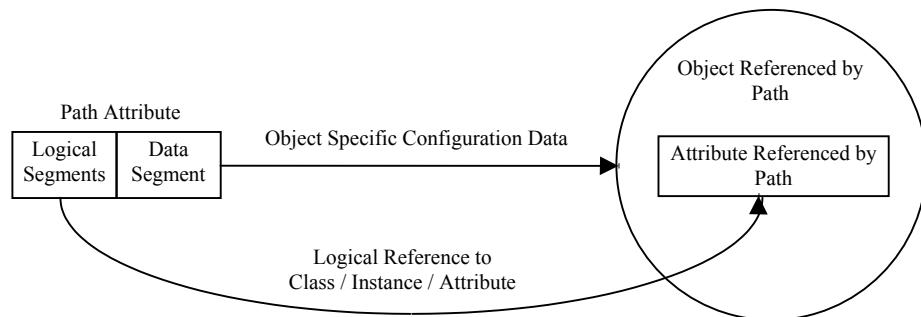
### C-1.2 Usage of Segments for Paths

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. 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:

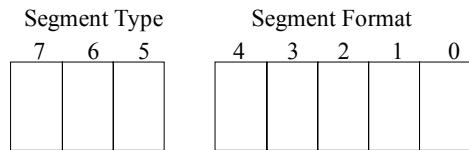
Padded Path (indicated as data type Padded EPATH)

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.



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 C-2.2)						
1    1    0	Data Type (elementary, see 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

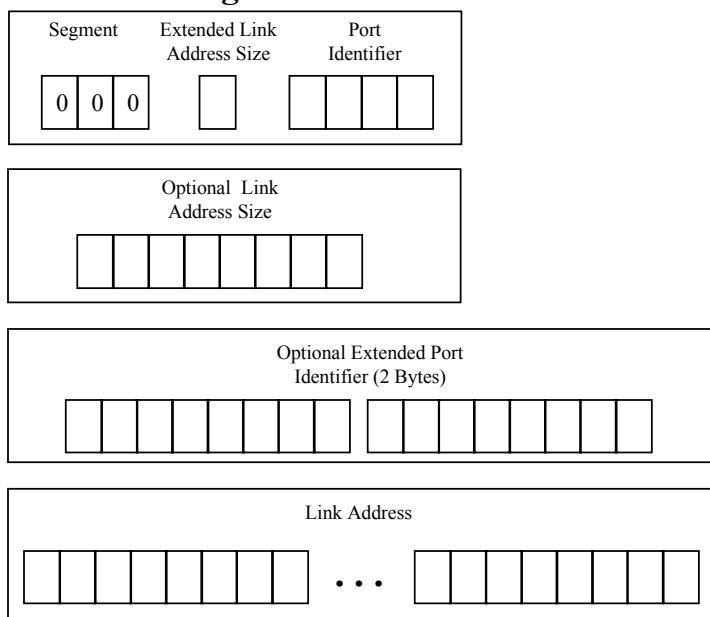
The port segment shall indicate

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.

## Port Segment



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).

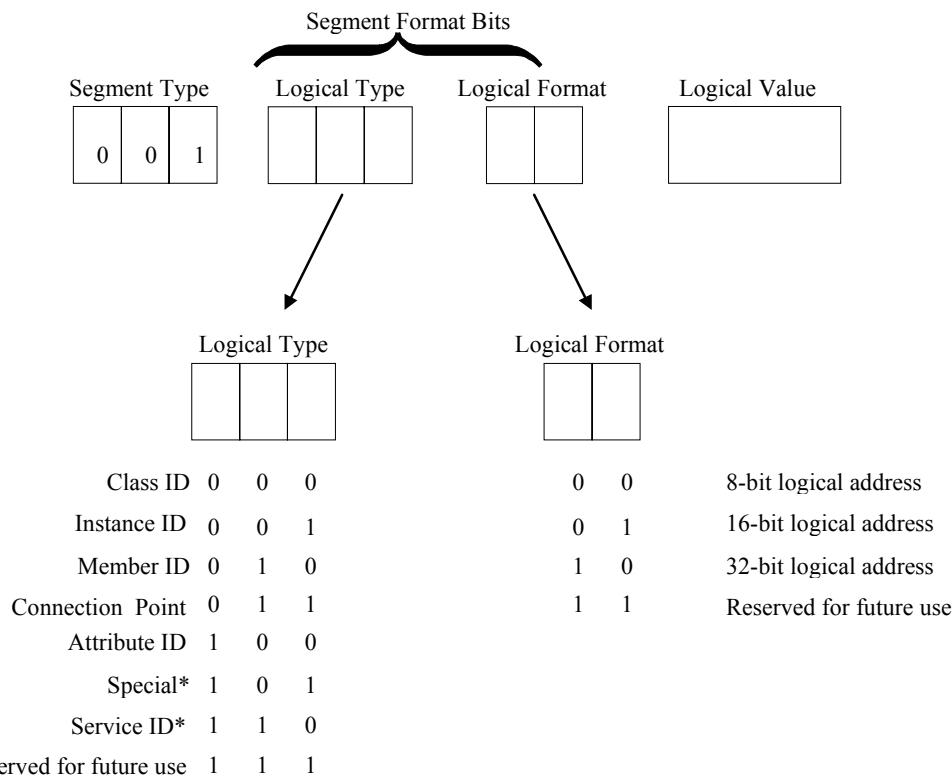
The Extended Port Identifier format of the Port Segment shall only be used when there are more than 14 network ports possible on the connecting device. The Port Segment shall always be represented in the smallest Port Segment format possible with respect to the optional fields.

**Examples** of Port Segments are presented below.

EPATH 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.



The *8-bit logical address* format is allowed for use with all Logical Types.

The *16-bit logical address* format is only allowed for use with Logical Types *Class ID*, *Instance ID*, *Member ID*, and *Connection Point*.

The *32-bit logical address* format is not allowed (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.

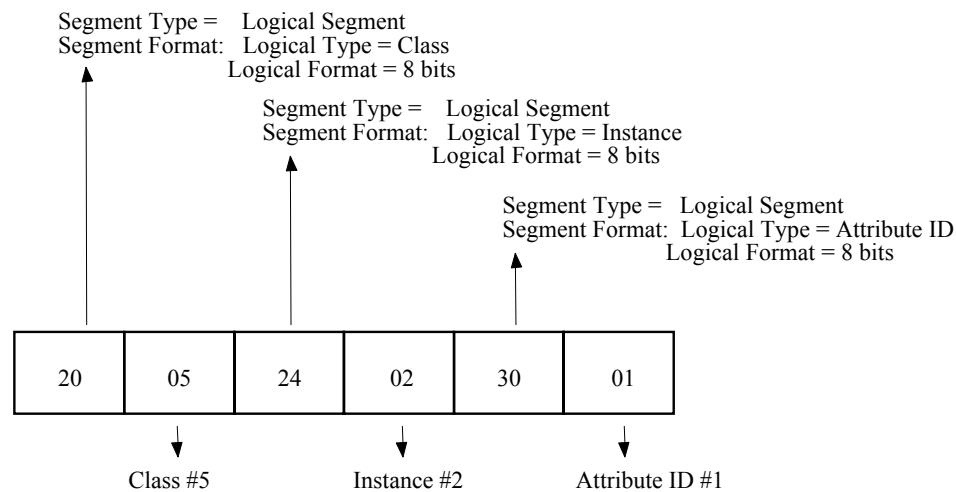
### **Electronic Key Segment Format**

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

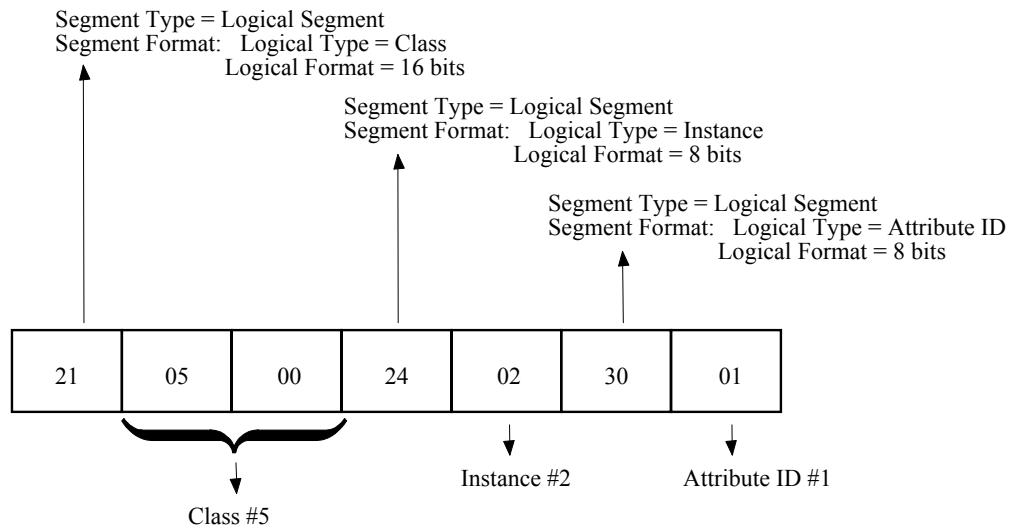
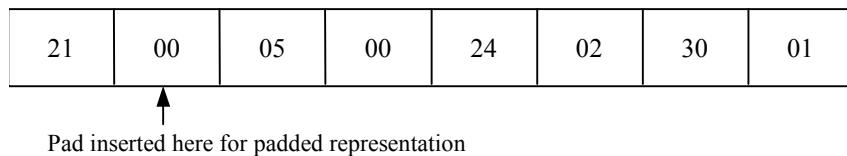
### Key Format Table

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):

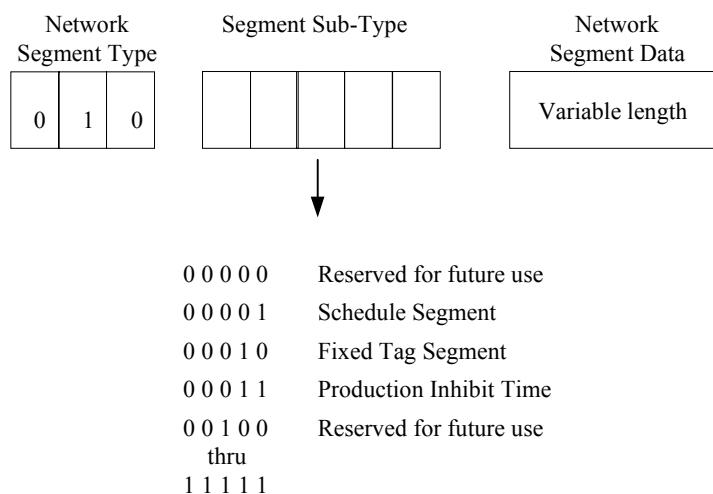


The same example is presented below, except that the Class information is specified in a 16 bit field. Figure C-1.2 shows a packed representation and Figure C-1.3 shows padded.

**Figure C-1.2 Packed EPATH with 16 bit Class****Figure C-1.3 Padded EPATH with 16 bit Class**

### 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.



### C-1.4.3.1 SCHEDULE NETWORK SEGMENT

The *Schedule* network segment is defined by the ControlNet International specification (Part 4, clause 3.2.2.4).

### C-1.4.3.2 FIXED TAG NETWORK SEGMENT

The *Fixed Tag* network segment is defined by the ControlNet International specification (Part 4, clause 3.2.2.4).

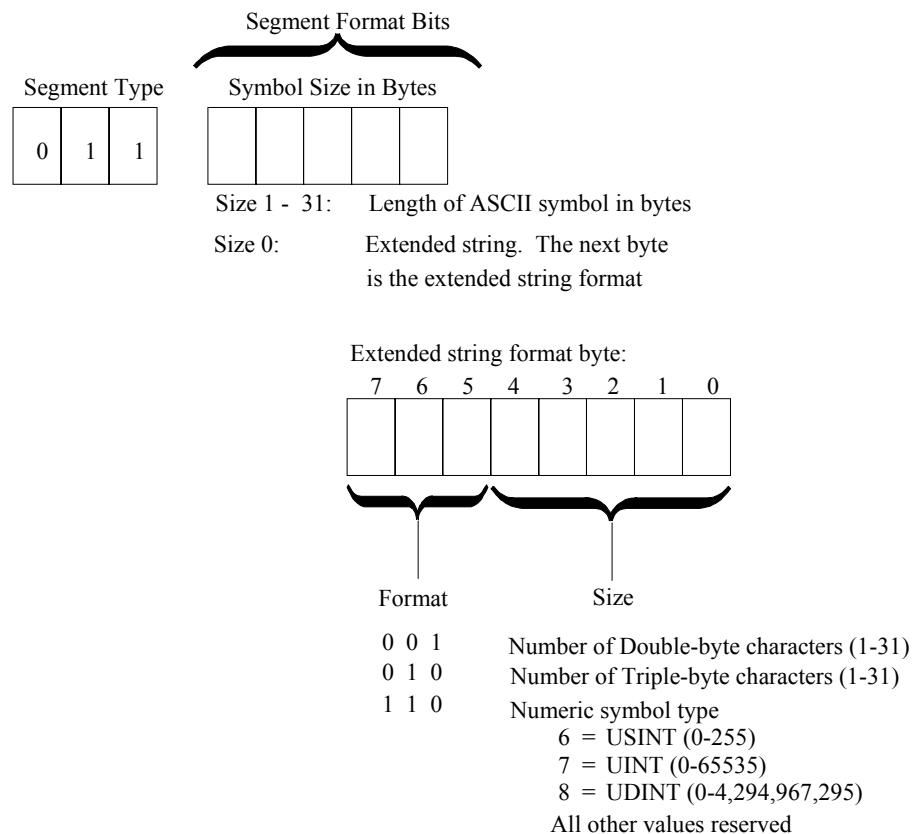
### C-1.4.3.3 PRODUCTION INHIBIT TIME NETWORK SEGMENT

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-1.4.4 SYMBOLIC SEGMENT

The symbolic segment contains an International String symbol which must be interpreted by the device.



Character Symbols can contain a maximum of 31 characters.

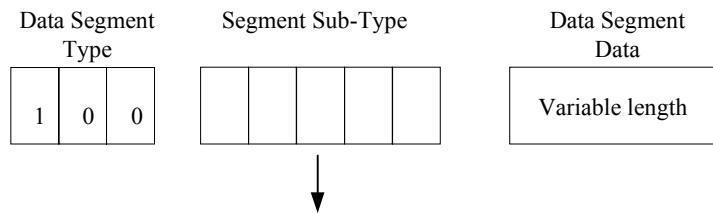
Numeric Symbols can be 8, 16, or 32 bits.

For example (values are hexadecimal):

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.



0 0 0 0 0	Simple Data Segment (0x80)
0 0 0 0 1	Reserved for future use
thru	
1 0 0 0 0	
1 0 0 0 1	ANSI Extended Symbol Segment (0x91)
1 0 0 1 0	Reserved for future use
thru	
1 1 1 1 1	

#### 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:

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:

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

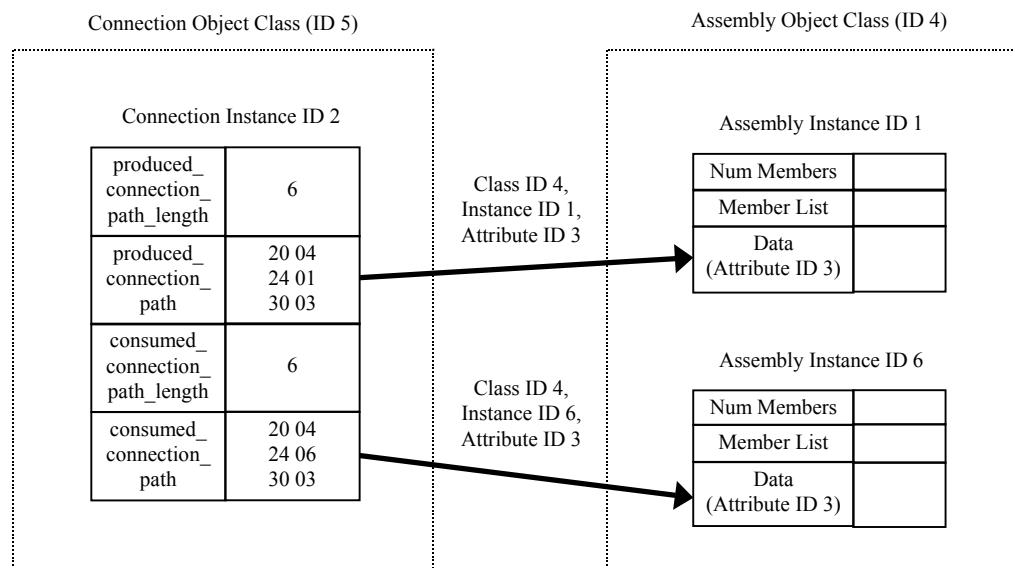
Class ID, Connection Point

Class ID, Connection Point, Member ID

Class ID, Instance ID, Connection Point

Class ID, Instance ID, Connection Point, Member ID

The following example demonstrates encoding for two path attributes: the **produced\_connection\_path** and **consumed\_connection\_path** attributes of the Connection Object.

**Figure C-1.4. Example Encoding for Connection Object Paths**

## 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	655351
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

Keyword	Description	Range	
		Minimum	Maximum
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
EPATH	CIP path segments		NOTE 13
ENGUNITS	ENGINEERING UNITS		NOTE 14
1	The possible values of variables of type BOOL 0 and 1, corresponding to the keywords FALSE and TRUE, respectively.		
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# $b_{N-1}b_{N-2}...b_2b_1b_0$ , where N is the number of bits in the bit string, $b_{N-1}$ is the “most significant bit”, and $b_0$ is the “least significant bit”. The value of the j-th bit $b_j$ 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.		

### 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.

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.

SHORT\_STRING, STRING2 and STRINGN are extensions to IEC 1131-3.

### 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-2.1.2 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
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

Table/ Feature	Feature description
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 J.4)
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 <b>NOTE— Table 30 of IEC 1131-3 limits the data types to which these operations apply.</b>
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
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> k <sup>2<sub>k</sub></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)
TIME_TO_ANY_NUM	(NOTE 16)	Result > OUT <sub>max</sub>
TOD_TO_ANY_NUM	(NOTE 17)	Result > OUT <sub>max</sub>

<sup>1</sup> 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.

<sup>2</sup> IN refers to the value of the input variable of the type conversion function.

<sup>3</sup> 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 J.1

Operation	Result	Error conditions
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 J.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 J.4
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 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 J.4 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 {
    charszie     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

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
}

```

```

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-4 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.

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-5.1. 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:

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-5.2. 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:

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-5.3. 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:

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  $\text{Float1:} = 10.0$ . (The assignment of the value is using the IEC 1131-3 notation. The ASN.1 value is  $\{\text{'41200000'H}\}$  in IEEE format ( $1.25*2^3$ , exponent is 130(bias 127), fraction is 25)).

**Table C-5.4. 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  $\text{Float2:} = -100.0$ . (  $\{\text{'C059000000000000'H}\}$  in IEEE format ( $1.5625*2^6$ , exponent is 1029 (bias 1023), fraction is .5625)).

**Table C-5.5. 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 FixedLengthReal values is given below:

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:

	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:

	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:

	Contents (charsize)		Contents (charcount)		Contents (stringNcontents)	
STRINGN	0LSB	.....	1LSB	0LSB	.....	1LSB

N byte character

A generic illustration of the encoding of a SHORT\_STRING value is given below:

	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-5.6. 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-5.7. Example Compact Encoding of String2 Value**

	Contents (charcount)		Contents (string2 contents)							
STRING2	04	00	4D	00	69	00	6C	00	6C	00

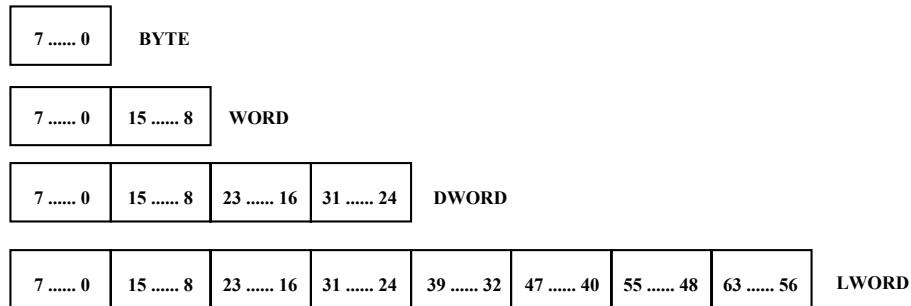
2 byte character

The example below encodes "Mill" as a SHORT\_STRING type.

	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 figure below illustrates the bit placement rules associated with the Compact Encoding of a FixedLengthBitString.



The examples below illustrate the encoding of a BYTE, WORD, DWORD, and an LWORD.

**Figure C-5.8. Example Compact Encoding of A BYTE FixedLengthBitString**

Bits In Memory:    7 ..... 0  
                      00001111

Compact Encoded BYTE  
00001111 or 0Fhex

**Figure C-5.9. Example Compact Encoding of A WORD FixedLengthBitString**

Bits In Memory:    15 ..... 0  
                      00001111 11001111

Compact Encoded WORD  
11001111 00001111 or CF0Fhex

**Figure C-5.10. Example Compact Encoding of A DWORD FixedLengthBitString**

Bits In Memory:    31 ..... 0  
                      00001111 11001111 10110110 00111110

Compact Encoded DWORD  
00111110 10110110 11001111 00001111 or 3EB6CF0Fhex

**Figure C-5.11. Example Compact Encoding of A LWORD FixedLengthBitString**

Bits In Memory:

63 .....	.....0
1110000 11001111 10110110 00111110 11110000 00101101 00011110 00001111	

Compact Encoded LWORD

00001111 00011110 00101101 11110000 00111110 10110110 11001111 11110000 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-5.12. 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-5.13. 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-5.14. 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
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

Data Type Name	Data Type Code (in hex)	Data Type Description
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

## 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 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							INT	
		UINT	Struct Type	Type Length	Nested Component					
					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 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 J-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 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
...							
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 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,
0xDBCl,      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,      0xBDCl,      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);  
}
```

## **Volume 1: CIP Common Specification**

## **Appendix D: Engineering Units**

---

This page is intentionally left blank

## 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 millamps, 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:

Value of MSB	Engineering Units Group
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

<b>Value of MSB</b>	<b>Engineering Units Group</b>
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 <sup>#2</sup>	s	
1101 hex	millisecond <sup>#3</sup>	ms	$10^{-3} \bullet s$
1102 hex	microsecond <sup>#4</sup>	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	$10^{-9} \bullet s$

<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.

Value	Name	Symbol	Base Units
1107 hex	DATE <sup>#5</sup>	D	
1108 hex	TIME_OF_DAY <sup>#6</sup>	TOD	
1109 hex	DATE_AND_TIME <sup>#7</sup>	DT	
110A-11FF hex	reserved for future standardization		

## D-2.5 Temperature

Value	Name	Symbol	Base Units
1200 hex	degree Celcius <sup>#8</sup>	°C	K - 273.15
1201 hex	degree Fahrenheit	°F	(K • 1.8) - 459.67
1202 hex	kelvin <sup>#9</sup>	K	
1203 hex	Degree Rankine	°R	K • 1.8
1204-12FF hex	reserved for future standardization		

## 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-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	

<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.

Value	Name	Symbol	Base Units
1402 hex	cubic foot per minute	CFM	ft <sup>3</sup> /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 <sup>-3</sup> • m <sup>3</sup> /s
1407 hex	milliliter per second	mL/s	10 <sup>-6</sup> • m <sup>3</sup> /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 <sup>-3</sup> • g/min
140F hex	grams per minute	g/M	g/min
1410 hex	kilograms per hour	kg/H	10 <sup>3</sup> • g/hr
1411-14FF hex	reserved for future standardization		

## D-2.8 Acceleration

Value	Name	Symbol	Base Units
1500 hex	meter per second squared	m/s <sup>2</sup>	
1501 hex	foot per second squared	ft/s <sup>2</sup>	
1502 hex	inch per second squared	in/s <sup>2</sup>	
1503 hex	angular acceleration	rad/s <sup>2</sup>	
1504 hex	acceleration during free fall (gravity)	g <sub>n</sub>	9.80665 • m/s <sup>2</sup>
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	mole per cubic meter	mol/m <sup>3</sup>	
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	

<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.

Value	Name	Symbol	Base Units
1702 hex	revolution	r	$6.283185 \bullet \text{rad}$
1703 hex	degree	°	$(\pi/180) \bullet \text{rad}$
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	m <sup>2</sup>	
1801 hex	square centimeter	cm <sup>2</sup>	
1802 hex	square kilometer	km <sup>2</sup>	
1803 hex	square yard	yd <sup>2</sup>	

Value	Name	Symbol	Base Units
1804 hex	square foot	ft <sup>2</sup>	
1804 hex	square foot	ft <sup>2</sup>	
1805 hex	square inch	in <sup>2</sup>	
1806 hex	square mile	mi <sup>2</sup>	
1807-18FF hex	reserved for future standardization		

## 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	μF	$10^{-6} \bullet \text{F}$
1903 hex	nano farad	nF	$10^{-9} \bullet \text{F}$
1904 hex	pico farad	pF	$10^{-12} \bullet \text{F}$
1905 hex	femto farad	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	μC	$10^{-6} \bullet \text{C}$
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

Value	Name	Symbol	Base Units
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	$10^{-1} \bullet \text{A}$
1C02 hex	milliamp	mA	$10^{-3} \bullet \text{A}$
1C03 hex	microamp	$\mu\text{A}$	$10^{-6} \bullet \text{A}$
1C04 hex	nanoamp	nA	$10^{-9} \bullet \text{A}$
1C05 hex	picoamp	pA	$10^{-12} \bullet \text{A}$
1C06 hex	femtoamp	fA	$10^{-15} \bullet \text{A}$
1C07 hex	kiloamp	kA	$10^3 \bullet \text{A}$
1C08 hex	megaamp	MA	$10^6 \bullet \text{A}$
1C09 hex	gigaamp	GA	$10^9 \bullet \text{A}$
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	$\text{m}^2 \bullet \text{kg} \bullet \text{s}^{-2}$
1D01 hex	watt second	W • s	J
1D02 hex	watt hour	W • h	$3.6 \bullet 10^3 \bullet \text{J}$
1D03 hex	kilowatt hour	kW • h	$3.6 \bullet 10^6 \bullet \text{J}$
1D04 hex	calorie (thermochemical, nutrition)	cal	$4.184 \bullet \text{J}$
1D05 hex	British thermal unit	Btu	$1005.056 \bullet \text{J}$
1D06 hex	kiloBtu	kBtu	$1005.056 \bullet 10^3 \bullet \text{J}$
1D07 hex	megaBtu	MBtu	$1005.056 \bullet 10^6 \bullet \text{J}$

<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.

Value	Name	Symbol	Base Units
1D08 hex	foot pound-force (foot-pound)	ft • lbf	$1.355818 \bullet J$
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		

## 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	

<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.

<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.

Value	Name	Symbol	Base Units
1F0E hex	revolution per second (rps)	rps <sup>#17</sup>	
1F0F hex	revolution per minute (rpm)	rpm <sup>#18</sup>	$1.047198 \bullet 10^{-1} \bullet \text{rad/s}$
1F10 hex	revolution per hour	r/h	
1F11 hex	revolution per day	r/d	
1F12-1FFF hex	reserved for future standardization		

## 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	nanohenry	nH	$10^{-9} \bullet \text{H}$
2004 hex	picoenry	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	$1.609344 \bullet 10^3 \bullet \text{m}$

<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.

<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.

Value	Name	Symbol	Base Units
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		

## 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	$10^{-3} \bullet \text{kg}$
2502 hex	milligram	mg	$10^{-6} \bullet \text{kg}$
2503 hex	metric ton (megagram)	t	$10^3 \bullet \text{kg}$
2504 hex	ounce (avoirdupois)	oz	$2.834952 \bullet 10^{-2} \bullet \text{kg}$
2505 hex	pound (avoirdupois)	lb	$4.535924 \bullet 10^{-1} \bullet \text{kg}$
2506 hex	short ton (2000 lb)		$9.071847 \bullet 10^2 \bullet \text{kg}$
2507-25FF hex	reserved for future standardization		

<sup>20</sup> When using the symbol for nautical mile (nm), care should be taken to avoid confusion with the symbol for nanometer.

<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		

## D-2.27 Resistance (Electric Resistance)

Value	Name	Symbol	Base Units
2800 hex	ohm	$\Omega$	$\text{m}^2 \bullet \text{kg} \bullet \text{s}^{-3} \bullet \text{A}^{-2}$
2801 hex	milliohm	$\text{m}\Omega$	$10^{-3} \bullet \Omega$
2802 hex	microohm	$\mu\Omega$	$10^{-6} \bullet \Omega$
2803 hex	nanoohm	$\text{n}\Omega$	$10^{-9} \bullet \Omega$
2804 hex	picoohm	$\text{p}\Omega$	$10^{-12} \bullet \Omega$
2805 hex	femtoohm	$\text{f}\Omega$	$10^{-15} \bullet \Omega$

<sup>23</sup> When there is risk of confusion with the symbol for the radian, rd may be used as the symbol for rad.

Value	Name	Symbol	Base Units
2806 hex	kilohm	$\text{k}\Omega$	$10^3 \bullet \Omega$
2807 hex	megaohm	$\text{M}\Omega$	$10^6 \bullet \Omega$
2808 hex	gigaohm	$\text{G}\Omega$	$10^9 \bullet \Omega$
2809 hex	ohm centimeter	$\Omega \bullet \text{cm}$	$10^{-2} \bullet \Omega \bullet \text{m}$
280A-28FF hex	reserved for future standardization		

## D-2.28 Sound

Value	Name	Symbol	Base Units
2900 hex	bel	B	
2901 hex	decibel	dB	$10^{-1} \bullet \text{B}$
2902 hex	Intensity ( $\text{W} \bullet \text{m}^{-2}$ )	I	
2903 hex	Sound Intensity Level	IL	see <sup>#24</sup>
2904 hex	Sound Pressure Level	SPL	see <sup>#25</sup>
2905 hex	Voltage Level re 1mW across 600Ω	dB(mW)	see <sup>#26</sup>
2906 hex	Voltage Level re 1 V	dB(V)	see <sup>#27</sup>
2907 hex	Voltage Level re 1 μV	dB(μV)	
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 • m	$\text{m}^2 \bullet \text{kg} \bullet \text{s}^{-2}$
2A01 hex	torque (mili)	$10^{-3} \bullet \text{N} \bullet \text{m}$	
2A02 hex	dyne centimeter	dyn • cm	$10^{-7} \bullet \text{N} \bullet \text{m}$
2A03 hex	kilogram-force meter	kgf • m	9.80665 • N • m
2A04 hex	ounce-force inch (ounce-inch)	ozf • in	$7.0615 \bullet 10^{-3} \bullet \text{N} \bullet \text{m}$
2A05 hex	pound-force inch (pound-inch)	lbf • in	$1.1298 \bullet 10^{-1} \bullet \text{N} \bullet \text{m}$
2A06 hex	pound-force foot (pound-foot)	lbf • ft	
2A07 hex	torque per ampere	(N • m)/A	
2A08 hex	torque per ampere (mili)	$10^{-3} \bullet ((\text{N} \cdot \text{m})/\text{A})$	
2A09-2AFF hex	reserved for future standardization		

<sup>24</sup> Sound Intensity Level is  $10 \bullet \log(\text{I}/\text{I}_0)$ , where  $\text{I}_0 = 10^{-12} \text{ W/m}^2$  in air and  $6.66 \cdot 10^{-19} \text{ W/m}^2$  in water.

<sup>25</sup> Sound Pressure Level is  $20 \bullet \log(\text{P}/\text{P}_0)$ , where  $\text{P}_0 = 20 \mu\text{Pa}$  in air and  $1 \mu\text{Pa}$  in water.

<sup>26</sup> Voltage Level re 1mW is  $20 \bullet \log(\text{V}/.775)$ , when the reference impedance is 600Ω.

<sup>27</sup> Voltage Level re 1 V is  $20 \bullet \log(\text{V}/1)$ , 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.)	$\text{gal}$	$\text{L} \bullet 3.785412 \bullet \text{L}$
2E09 hex	quart (U.S. liquid)	$\text{liq qt}$	$9.463529 \bullet 10^{-1} \bullet \text{L}$
2E0A hex	pint (U.S. liquid)	$\text{pt}$	$4.731765 \bullet 10^{-1} \bullet \text{L}$
2E0B hex	ounce (U.S. fluid)	$\text{fl oz}$	$2.957353 \bullet 10^1 \bullet \text{mL}$
2E0C hex	barrel (U.S.)	$\text{bbl}$	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
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-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.

This page is intentionally left blank