CDN36X Series
DeviceNet Gateway
User Manual

CDN366 – 1 isolated RS232 channel
CDN367 – 1 isolated RS422/RS485 channel

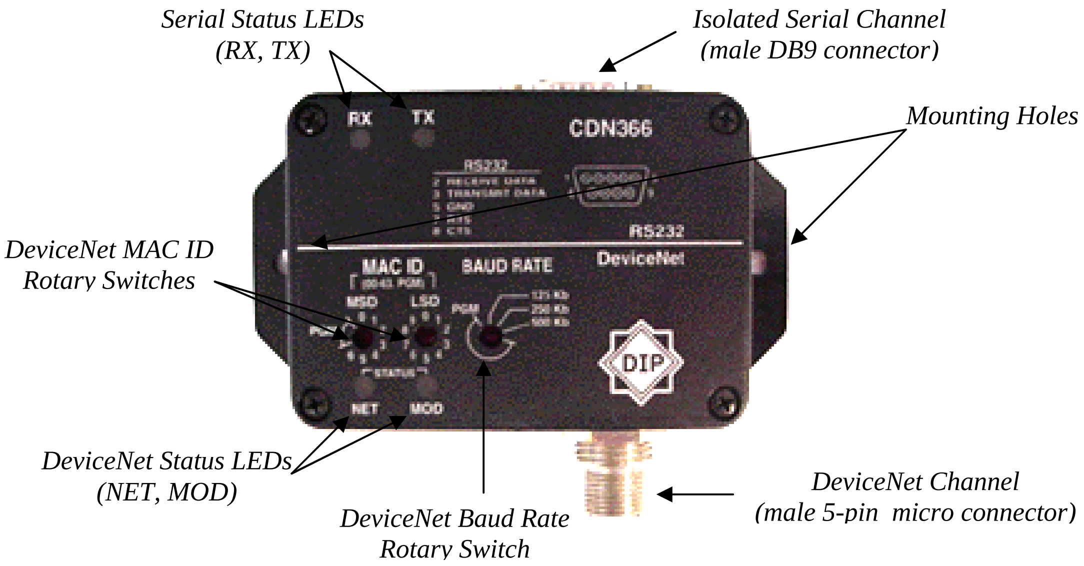
Table of Contents

Chapter 1 – Overview

This document describes how to install, configure, and operate the CDN36X series of serial to DeviceNet gateways. The following products are covered in this user manual:

Part Number	FW Rev.	Serial Channel
CDN366	2.04 or higher	RS232 full duplex
CDN367	2.04 or higher	RS422 full duplex / RS485 half duplex

The CDN36X gateways allow you to easily interface a wide variety of serial devices to any DeviceNet industrial control network. Each gateway contains the feature-packed D.I.P. DeviceNet core. Standard CDN36X products are tightly packaged and sealed in a rugged industrial case. Board-level and customized gateways are also available upon request.


Serial Status LEDs (RX, TX)
Isolated Serial Channel (male DB9 connector)
Mounting Holes
DeviceNet MAC ID Rotary Switches
DeviceNet Status LEDs (NET, MOD)
DeviceNet Baud Rate Rotary Switch
DeviceNet Channel (male 5-pin micro connector)
RX
TX
CDN366
RS232
MAC ID
BAUD RATE
DeviceNet
NET
MOD
DIP

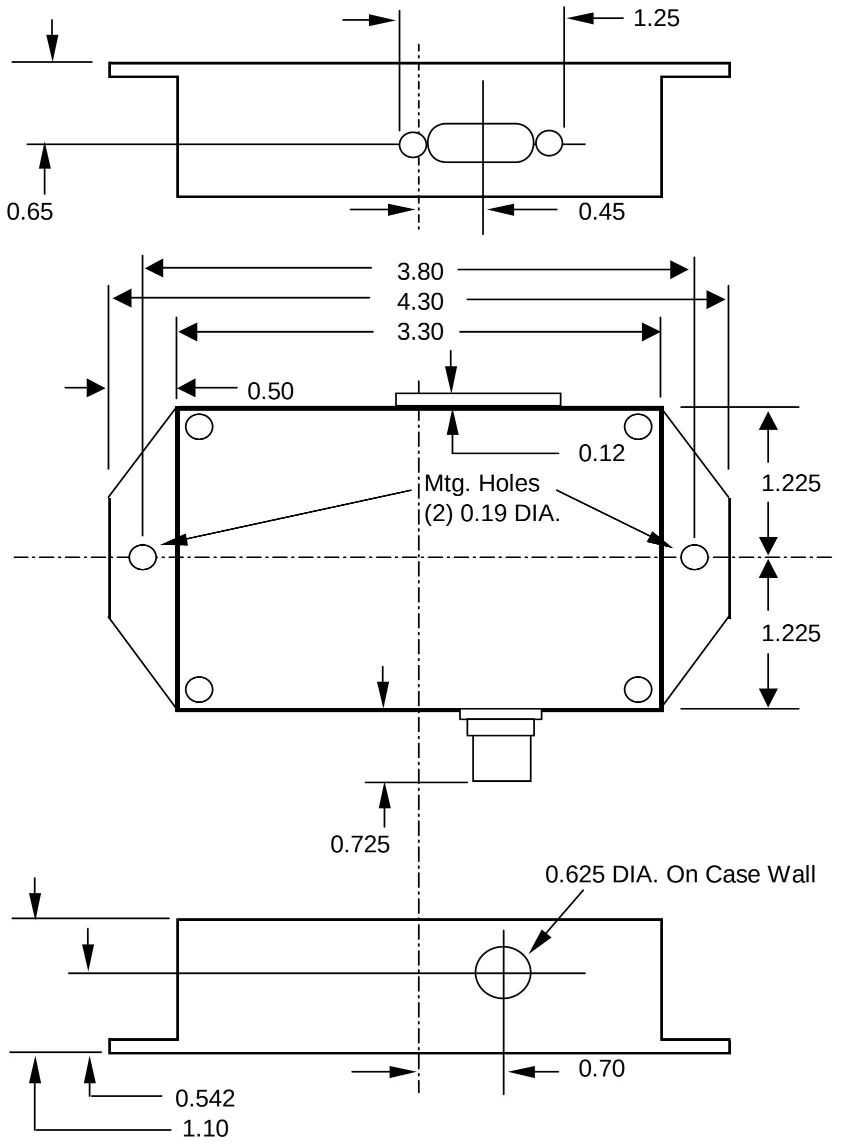
Product Features	**CDN366**	**CDN367**
• 500V isolated serial channel	X	X
• RS232 with RTS/CTS flow control	X	
• RS422 full duplex (4-wire) with terminating resistors		X
• RS485 half duplex (2-wire) with terminating resistor, repeater control signal		X
• XON/XOFF software flow control	X	X
• 300, 1200, 2400, 4800, 9600, 19200 bps serial data rates	X	X
• Configurable data bits, stop bits, parity	X	X
• 128 byte transmit and 128 receive FIFO buffers	X	X
• Receives up to 8 different serial messages	X	X
• Transmits up to 8 different serial messages	X	X
• Powered from DeviceNet 24VDC	X	X
• Loss-of-ground protection circuitry	X	X
• DeviceNet slave mode supports POLL, COS, EXPLICIT messages	X	X
• Rotary switches set DeviceNet baud rate and MAC ID	X	X
• 4 bi-color status LEDs	X	X
• I/O Byte-swap I/O option for compatibility with PLC Scanners	X	X

Chapter 2 – Installation

This chapter describes how to install and connect the CDN36X gateway to a DeviceNet network and your serial device.

Mounting

Mount on a horizontal or vertical surface. While the RTV encapsulation protects its circuitry, the CDN36X serial channel connector is not rated for NEMA4 / IP65 environments. Mount the gateway in a suitable location or enclosure for your application. The gateway will generate up to 1.4W of heat, so provide sufficient clearance and airflow to maintain 0°C to 70°C operating temperature range. Use two screws (not provided) in the 0.19 inch mounting holes shown below to fasten the CDN36X to the mounting surface.



All dimensions are inches

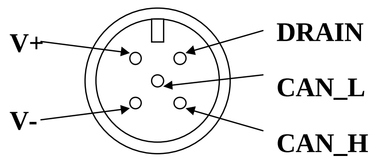
Wiring

The CDN36X requires two connections – one to the DeviceNet network (male 5-pin micro connector) and one to the target serial device (male DB9 connector). Follow all applicable electrical codes in your area when mounting and wiring any electrical device.

All power is received from the DeviceNet network. The CDN36X draws up to 50mA from the 24VDC power supply. Select your DeviceNet cables and power supply so that it can provide sufficient current for all networked devices at their peak operating power.

DeviceNet Interface

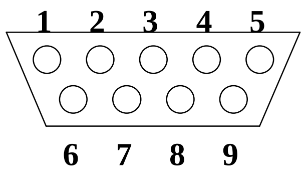
Male 5-Pin Micro Connector



PIN	SIGNAL	COLOR	DESCRIPTION
1	DRAIN	NONE	Cable shield or drain wire.
2	V+	RED	DeviceNet 24VDC(+) power.
3	V-	BLACK	DeviceNet 24VDC(-) power.
4	CAN_H	WHITE	Communication signal.
5	CAN_L	BLUE	Communication signal.

Serial Channel Interface

Male DB9 Serial Connector



CDN366 (RS232)

PIN	SIGNAL	DESCRIPTION
1	NC	No Connect. Do not connect any wires to NC pins.
2	RXD	Receive Data. RS232 input signal.
3	TXD	Transmit Data. RS232 output signal.
4	NC	No Connect.
5	GND	Ground. Common for RS232 signals.
6	NC	No Connect.
7	RTS	Request To Send. RS232 output signal.
8	CTS	Clear To Send. RS232 input signal.
9	NC	No Connect.

CDN367 (2-WIRE RS485 configuration)

PIN	SIGNAL	DESCRIPTION
1	RXA	RS485 differential data I/O signal.
2	RXB	RS485 differential data I/O signal.
3	TR	Internal 120Ω Terminating Resistor, connected between pins 1 and 3. Connect pin 2 to pin 3 to terminate DATA signals. Use at end of long twisted-pair cable.
4	NC	No Connect. Do not connect any wires to NC pins.
5	GND	Ground.
6	TXA	Connect to pin 1 for 2-wire operation.
7	TXB	Connect to pin 2 for 2-wire operation.
8	TR2	Internal 120Ω Terminating Resistor, connected between pins 6 and 8. Do not connect in 2-wire operation.
9	+5VDC	Auxiliary 5VDC supply generated by CDN36X.

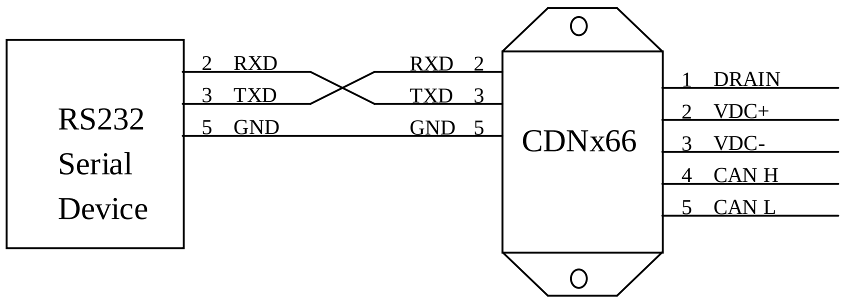
CDN367 (4-WIRE RS422 configuration)

PIN	SIGNAL	DESCRIPTION
1	RXA	RS422 differential receive data input signal.
2	RXB	RS485 differential receive data input signal.
3	TR	Internal 120Ω Terminating Resistor, connected between pins 1 and 3. Connect pin 3 to pin 2 to terminal RX signals. Use at end of long twisted-pair cable.
4	NC	No Connect. Do not connect any wires to NC pins.
5	GND	Ground.
6	TXA	RS422 differential transmit data output signal.
7	TXB	RS422 differential transmit data output signal.
8	TR2	Internal 120Ω Terminating Resistor, connected between pins 6 and 8. Connect pin 7 to pin 8 to terminate TX signals. Use at end of long twisted-pair cable.
9	+5VDC	Auxiliary 5VDC supply generated by CDN36X.

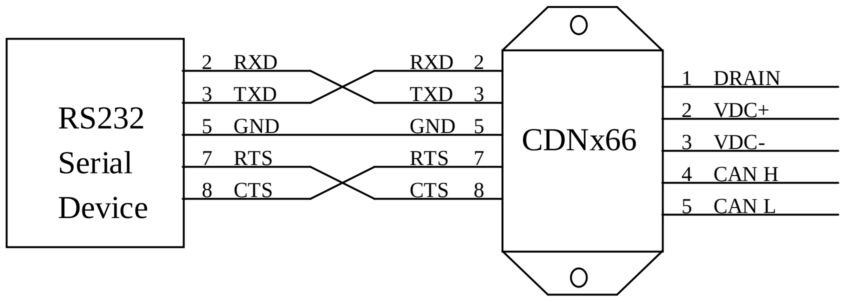
Wiring Examples

The following are typical CDN36X gateway wiring configurations. Your RS232 or RS422/485 interface may vary. Refer to your device's documentation for the required data and control signals.

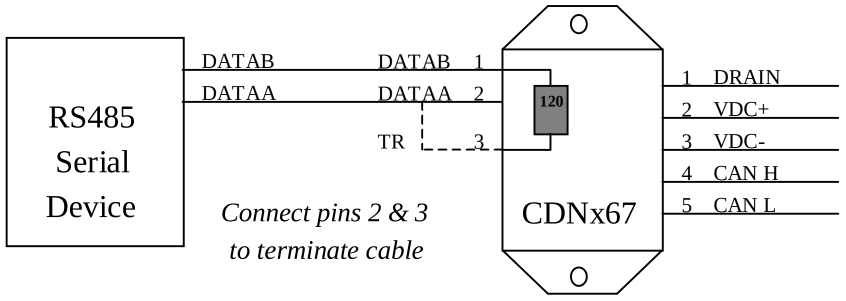
Simple RS232 Interface



RS232 Interface, HW Flow Control

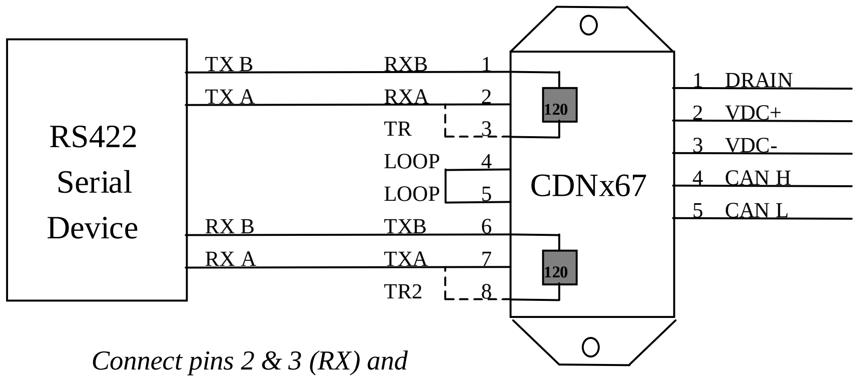


Simple RS485 Interface



Connect pins 2 & 3 to terminate cable

RS422 4-Wire Interface



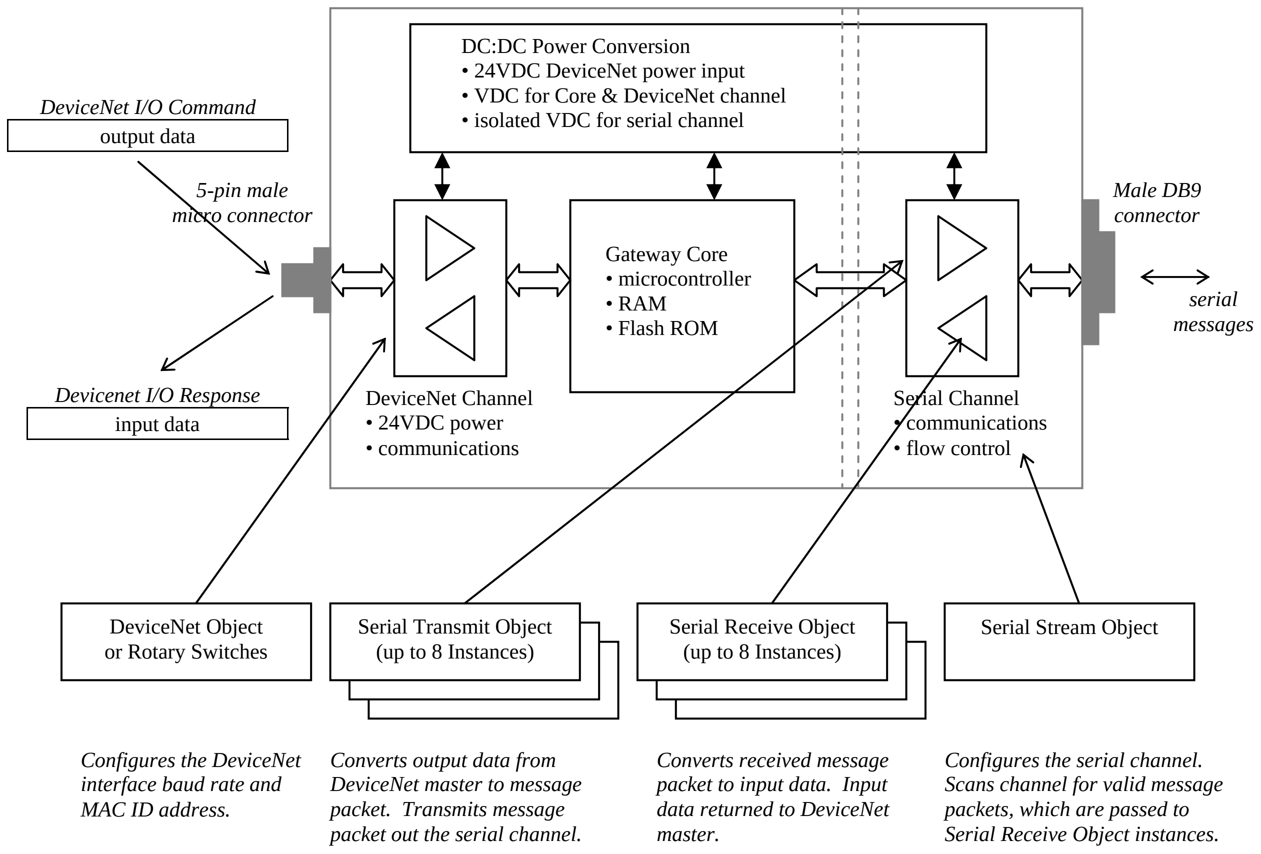
Connect pins 2 & 3 (RX) and pins 7 & 8 (TX) to terminate cables. Connect pins 4 & 5.

Chapter 3 – Theory of Operation

This chapter describes how the CDN36X gateway operates. You should have a working knowledge of DeviceNet and asynchronous serial communications before continuing. The Open DeviceNet Vendors Association (www.odva.com) is a good source for general DeviceNet information. Refer to your serial device documentation for its protocol information.

Gateway Operation

The CDN36X gateway receives asynchronous serial messages over its serial channel, converts them to data values, and returns the values as input data to the DeviceNet master. The gateway receives output data from the DeviceNet master, converts them into serial messages, and transmits the messages out its serial channel. The following diagram shows the major gateway components.


DC:DC Power Conversion
• 24VDC DeviceNet power input
• VDC for Core & DeviceNet channel
• isolated VDC for serial channel
DeviceNet I/O Command
output data
5-pin male micro connector
Devicenet I/O Response
input data
Gateway Core
• microcontroller
• RAM
• Flash ROM
DeviceNet Channel
• 24VDC power
• communications
Serial Channel
• communications
• flow control
Male DB9 connector
serial messages
DeviceNet Object or Rotary Switches
Serial Transmit Object (up to 8 Instances)
Serial Receive Object (up to 8 Instances)
Serial Stream Object
Configures the DeviceNet interface baud rate and MAC ID address.
Converts output data from DeviceNet master to message packet. Transmits message packet out the serial channel.
Converts received message packet to input data. Input data returned to DeviceNet master.
Configures the serial channel. Scans channel for valid message packets, which are passed to Serial Receive Object instances.

DeviceNet Object Model

The DeviceNet Specification defines an Object Model that consists of Objects and Attributes. An Object is a predefined software process, and an Object Attribute is a data value used or created by that process. An Object can have multiple Instances, or the same process operating with different sets of Attributes or data values. For the purpose of this document, an Object Instance is an independent program or process, and its Attributes are configuration parameters and data values that are unique to that specific Object Instance.

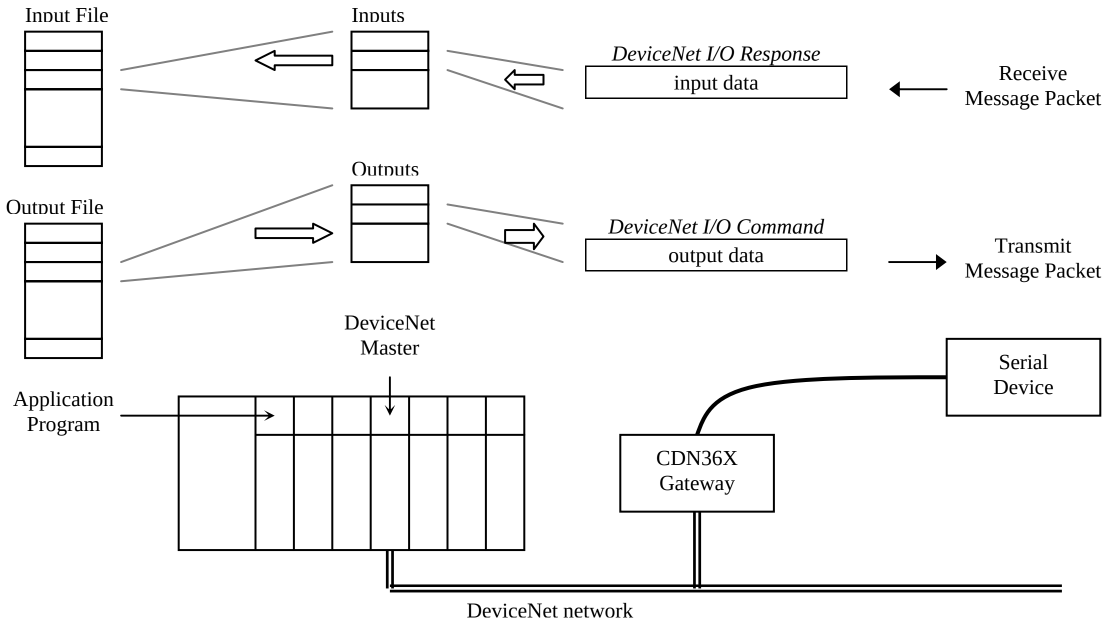
The CDN36X gateway has eight different Object Classes, or types. Five are standard objects defined by the DeviceNet Specification (*Identity, Router, DeviceNet, Assembly, Connection*). Three are specific objects defines for the CDN36X gateway (*Serial Stream, Serial Receive, Serial Transmit*). The *Serial Stream Object* configures the serial channel, and scans the incoming serial stream for valid message packets. The *Serial Receive Object* processes the received message packet, converts it into input data, and returns it to the DeviceNet master. The *Serial Transmit Object* receives output data from the DeviceNet master, converts it into a message packet, and transmits it out the serial channel. You can configure up to 8 Instances each for the *Serial Receive Object* and *Serial Transmit Object*. Chapter 5 contains detailed information on each DeviceNet object class.

DeviceNet Interface

The CDN36X gateway operates as a DeviceNet slave. It supports Explicit Messages, Polled I/O Messages, and Change-of-State (COS) I/O Messages of the predefined master/slave connection set. The Explicit Unconnected Message Manager (UCMM) is not supported.

The I/O Messaging process consists of the DeviceNet master sending output data to the CDN36X in the form of a Poll/COS Command Message, and the CDN36X returning input data to the DeviceNet master in a Poll/COS Response Message. The difference between Poll and Change-of-State is Polled I/O Messaging is initiated by the DeviceNet master and responded to by the slave device. Change-of-State I/O Messaging is initiated by changes to the master's output data values or the slave's input data values, causing the master or slave to immediately transmit its new output or input data when it changes. Please refer to DeviceNet Specification for detailed information on Polled I/O and Change-of-State I/O Messaging.

The output and input data bytes are typically mapped into data files inside the DeviceNet master. These data files are exchanged with the user application program, which acts upon the received input data and writes new output data to the DeviceNet master.



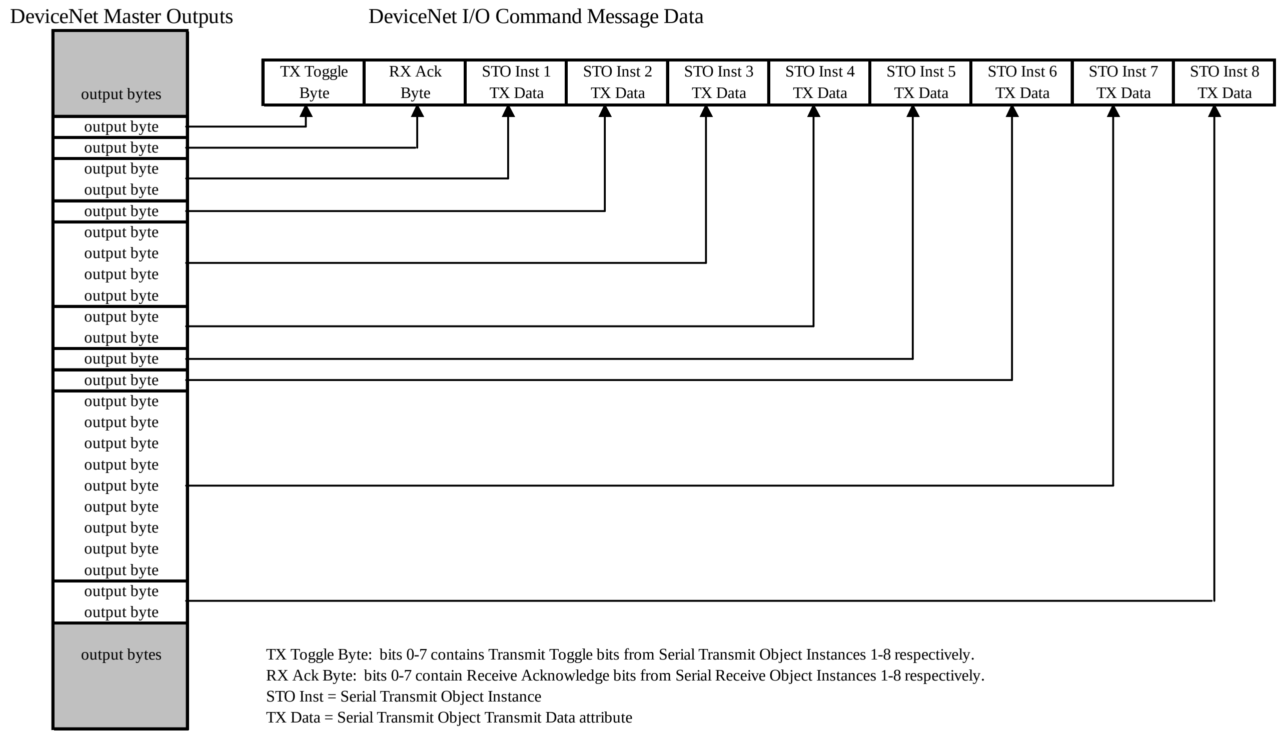
The first 2 output data bytes received from the DeviceNet master contain synchronization bits for the gateway transmit and receive operations. The remaining output data bytes contain serial message data to be transmitted out the serial channel. Up to 8 different output data values can be sent in an I/O Command Message, one for each enabled *Serial Transmit Object* Instance.

The first 2 input data bytes sent from the gateway contain synchronization bits for the gateway transmit and receive operations. The remaining input data bytes contain serial message data that has been received and processed by the gateway. Up to 8 different input data values can be returned in an I/O Response Message, one for each enabled *Serial Receive Object* Instance.

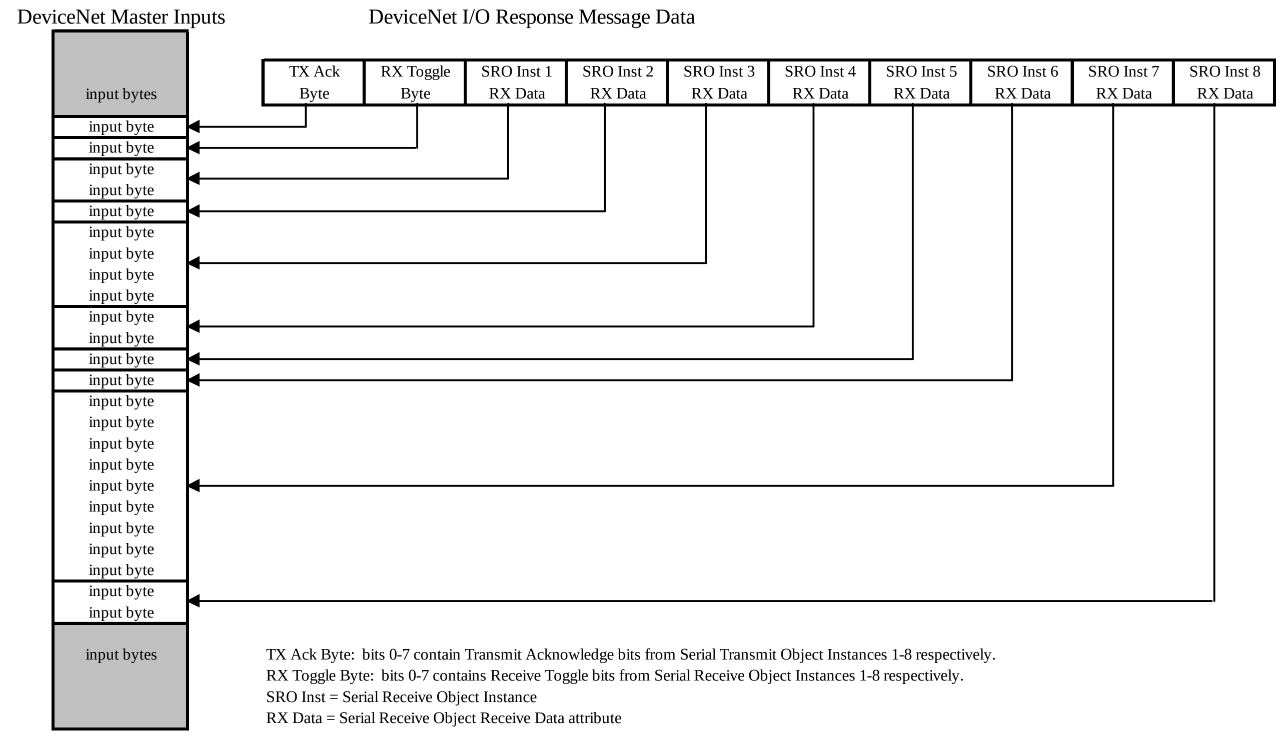
The following diagram shows how the gateway's input and output data bytes map into the DeviceNet I/O Response and I/O Command Messages. The diagram includes all 8 *Serial Receive Object* Instances and 8 *Serial Transmit Object* Instances. If an instance is not enabled, its data bytes are not mapped or present in the corresponding I/O Message. The total number of input or output data bytes required for each Instance is defined by its configuration.

The gateway supports a maximum of 128 data bytes for all 8 *Serial Receive Object* Instances and 128 bytes for all 8 *Serial Transmit Object* Instances, regardless of whether an instance is enabled or not. Unused instances should have their data size set to the smallest number of bytes.

DeviceNet Master Mapping of DeviceNet I/O Command and I/O Response Data



TX Toggle Byte: bits 0-7 contains Transmit Toggle bits from Serial Transmit Object Instances 1-8 respectively.
RX Ack Byte: bits 0-7 contain Receive Acknowledge bits from Serial Receive Object Instances 1-8 respectively.
STO Inst = Serial Transmit Object Instance
TX Data = Serial Transmit Object Transmit Data attribute



TX Ack Byte: bits 0-7 contain Transmit Acknowledge bits from Serial Transmit Object Instances 1-8 respectively.
RX Toggle Byte: bits 0-7 contains Receive Toggle bits from Serial Receive Object Instances 1-8 respectively.
SRO Inst = Serial Receive Object Instance
RX Data = Serial Receive Object Receive Data attribute

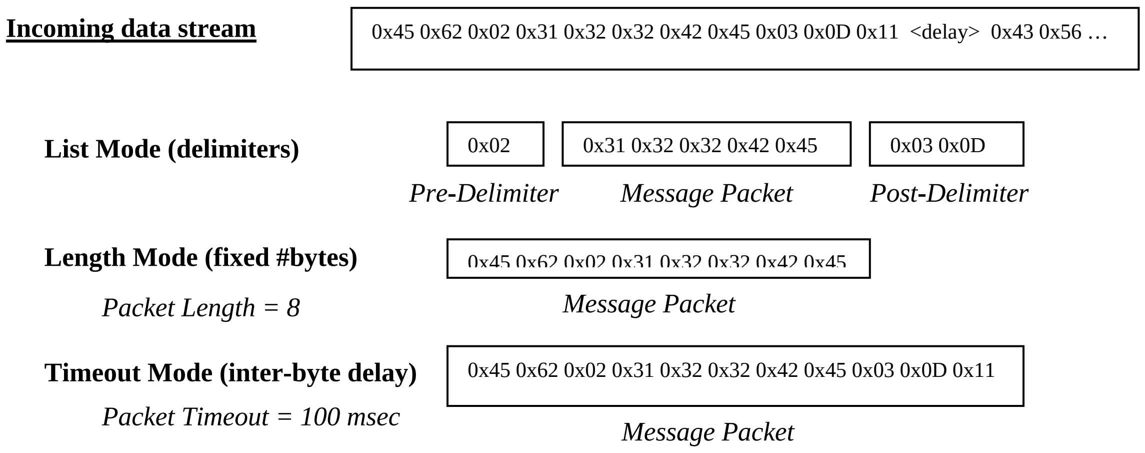
Serial Channel Interface

The CDN36X serial channel consists of an asynchronous serial transmitter and receiver. The serial interface is configured and controlled by the *Serial Stream Object, Serial Receive Object,* and *Serial Transmit Object*.

Serial Stream Object

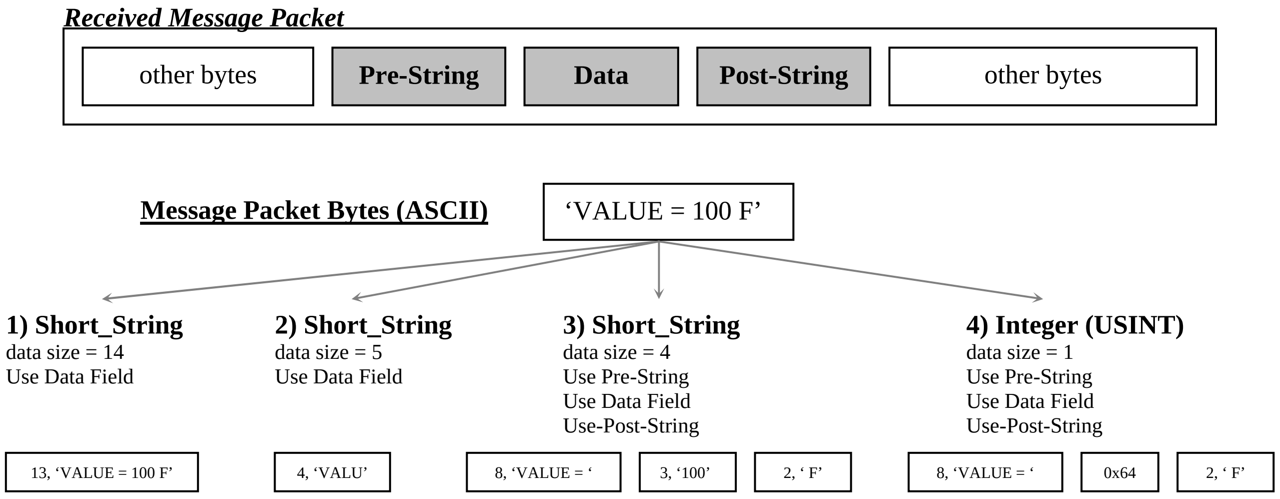
The *Serial Stream Object* attributes configure the serial channel's baud rate, number of data bits and stop bits, parity, and flow control. This configuration applies to both the serial transmitter and receiver. The gateway has separate 128-byte serial transmit and receive FIFO buffers, allowing full duplex operation when supported by the physical layer media.

The *Serial Stream Object* also scans incoming serial data for valid message packets. A message packet is determined by one of three *Delimiter* modes. *List* mode searches for *Pre-Delimiter* and *Post-Delimiter* byte strings at the beginning and end of a message. *Length* mode captures a specific number of message bytes, defined by *Packet Length*. *Timeout* mode uses an inter-byte delay (*Packet Timeout*) to signal the end of a message. When a message packet is received, it is processed by all enabled *Serial Receive Object* Instances. The following examples show the three *Serial Stream Object Delimiter* modes.

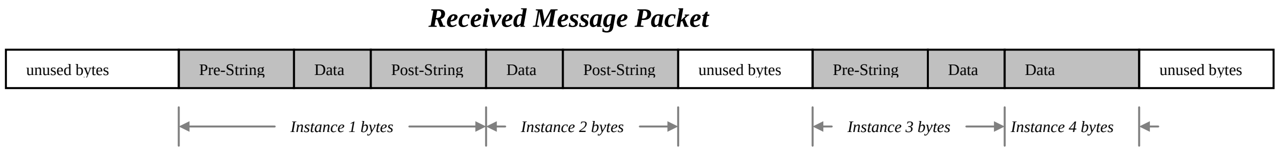

Incoming data stream
0x45 0x62 0x02 0x31 0x32 0x32 0x42 0x45 0x03 0x0D 0x11 <delay> 0x43 0x56 …
List Mode (delimiters)
0x02
0x31 0x32 0x32 0x42 0x45
0x03 0x0D
Pre-Delimiter
Message Packet
Post-Delimiter
Length Mode (fixed #bytes)
Packet Length = 8
0x45 0x62 0x02 0x31 0x32 0x32 0x42 0x45
Message Packet
Timeout Mode (inter-byte delay)
Packet Timeout = 100 msec
0x45 0x62 0x02 0x31 0x32 0x32 0x42 0x45 0x03 0x0D 0x11
Message Packet

Serial Receive Object

The *Serial Receive Object* processes the Message Packet bytes, converting them into an input data value that is returned to the DeviceNet master in an I/O Response Message. The Message Packet bytes can be converted into a *Short_String* data type (byte array, with 1st byte = length). ASCII characters within the Message Packet representing a numerical value can also be converted into signed or unsigned integer or real number data types. The *Serial Receive Object* can be configured to search for *Pre-String* and/or *Post-String* byte strings at the beginning and/or end of the desired data bytes. The data bytes framed by the *Pre-String* and *Post-String* bytes are then converted into a Short_String, integer, or real number. The following examples show how the *Serial Receive Object* can be configured to process a Message Packet.

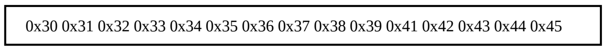


You can configure up to 8 *Serial Receive Object* Instances, allowing you to process a received message packet against 8 different filters. Each enabled Instance parses through the message packet, consuming bytes based on its configuration. When an Instance is finished, the remaining message packet bytes are passed to the next enabled Instance. The *Serial Receive Object* sequentially processes a message packet in order of Instance number, starting with Instance 1. The processing continues until either all message packet bytes have been consumed or all the enabled Instances are done. If an Instance cannot find a string of message bytes that matches its configuration, no message packet bytes are consumed and are passed to the next Instance. The following examples show how the *Serial Receive Object* Instances process a received message packet.

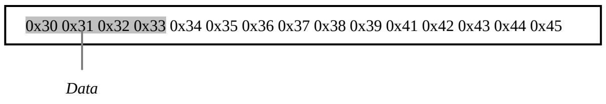


<u>Example 1</u>
Serial Receive Object instances 1, 4, 5, 7 are configured to use portions of a received message packet.

1) *Serial Stream Object* configured to capture a 15 byte message packet (Length Mode, Packet Length = 15).

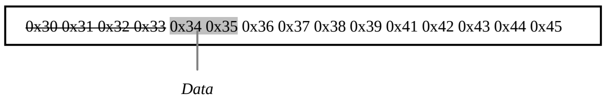

0x30 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0x39 0x41 0x42 0x43 0x44 0x45

2) *Serial Receive Object* Instance 1 configured to convert 4 bytes into Short_String, Data Size = 5. Short_String = [0x05 0x30 0x31 0x32 0x33].

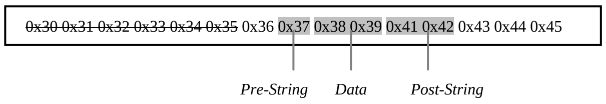

0x30 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0x39 0x41 0x42 0x43 0x44 0x45
Data

3) *Serial Receive Object* Instances 2 & 3 not enabled.

4) *Serial Receive Object* Instance 4 configured to convert 2 bytes into 8-bit unsigned integer. [0x34 0x35] are ASCII chars '45'. Converted data = [0x2D].

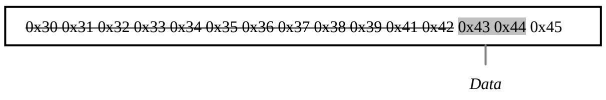

~~0x30 0x31 0x32 0x33~~ 0x34 0x35 0x36 0x37 0x38 0x39 0x41 0x42 0x43 0x44 0x45
Data

5) *Serial Receive Object* Instance 5 configured to locate 2 bytes bracketed by Pre-String '7' (0x01 0x37) and Post-String 'AB' (0x02 0x41 0x42), and convert them to 8-bit unsigned integer. [0x38 0x39] are ASCII chars '89'. Converted data = [0x59]. Note 0x36 byte not used, but still consumed by Instance 5.


~~0x30 0x31 0x32 0x33 0x34 0x35~~ 0x36 0x37 0x38 0x39 0x41 0x42 0x43 0x44 0x45
Pre-String
Data
Post-String

6) *Serial Receive Object* Instance 6 not enabled.

7) *Serial Receive Object* Instance 7 configured to convert 2 bytes into Short_String, Data Size = 3. Short_String = [0x02 0x43 0x44].


~~0x30 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0x39 0x41 0x42~~ 0x43 0x44 0x45
Data

8) Remaining message packet bytes are not used.

Example 2
Serial Receive Object Instances 1 & 2 are configured to process two different ASCII message strings. Each instance uses unique Pre-String and Post-String values to identify its message string in the message packet. Instance 1 message string = ‘VALUE = xxx U’. Instance 2 message string = ‘VAR B IS xxx’.

Serial Stream Object configured to capture a message packet delimited by STX (0x02) and ETX (0x03) characters. Delimiter Mode = List Mode, Pre-Delimiter = 0x02, Post-Delimiter = 0x03. Delimiter characters are not saved in the message packet.

Serial Receive Object Instance 1 configured to convert 3 ASCII bytes bracketed by Pre-String ‘VALUE = ‘ and Post-String ‘ U’ into an 8-bit unsigned integer. Receive Mode = Use Pre-String, Use Data, Use Post-String fields. Pre-String = [0x08, ‘VALUE = ‘], Post-String = [0x02, ‘ U’], Data Type = USINT, Conversion = Decimal.

Serial Receive Object Instance 2 configured to convert 3 ASCII bytes bracketed by Pre-String ‘VAR B IS ‘ into an 8-bit unsigned integer. Receive Mode = Use Pre-String, Use Data fields. Pre-String = [0x09 ‘VAR B IS ‘], Data Type = USINT, Conversion = Decimal.

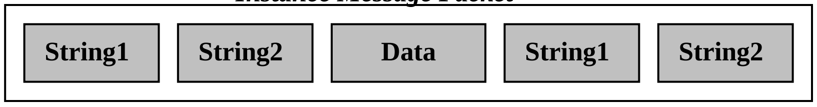
Message Packet 1 = [0x02, ‘VALUE = xxx UNITS’, 0x03]
Message Packet 2 = [0x02, ‘VAR B IS xxx’, 0x03]
Message Packet 3 = [0x02, ‘VALUE = xxx UNITS’, ‘VAR B IS xxx’, 0x03]
Message Packet 4 = [0x02, ‘VAR B IS xxx’, ‘VALUE = xxx UNITS’, 0x03]

1) *Serial Stream Object* receives Message Packet 1.	‘VALUE = 100 UNITS’
Serial Receive Object Instance 1 finds matching Pre-String and Post-String values, converts ‘100’ data field into [0x64].	‘VALUE = 100 UNITS’
Serial Receive Object Instance 2 does not find matching Pre-String field, so it skips this message packet.	‘~~VALUE = 100 U~~NITS’
2) *Serial Stream Object* receives Message Packet 2.	‘VAR B IS 104’
Serial Receive Object Instance 1 does not find matching Pre-String and Post-String values, so it skips this message packet.	‘VAR B IS 104’
Serial Receive Object Instance 2 finds matching Pre-String value, converts ‘104’ data field into [0x68].	‘VAR B IS 104’
3) *Serial Stream Object* receives Message Packet 3.	‘VALUE = 122 UNITSVAR B IS 080’
Serial Receive Object Instance 1 finds matching Pre-String and Post-String values, converts ‘122’ data field into [0x7A].	‘VALUE = 122 UNITSVAR B IS 080’
Serial Receive Object Instance 2 finds matching Pre-String value, converts ‘080’ data field into [0x50]. Note that Instance 2 consumes the bytes ‘NITS’ while parsing for its Pre-String.	‘~~VALUE = 122 U~~NITSVAR B IS 080’
4) *Serial Stream Object* receives Message Packet 4.	‘VAR B IS 080VALUE = 122 UNITS’
Serial Receive Object Instance 1 finds matching Pre-String and Post-String values, converts ‘122’ data field into [0x7A].	‘VAR B IS 080VALUE = 122 UNITS’
Serial Receive Object Instance 2 does not find matching Pre-String value, so skips this message packet. Note that Instance 1 consumes the Instance 2 message string bytes when parsing for its message string.	‘~~VAR B IS 080VALUE = 122 U~~NITS’

<u>*Serial Transmit Object*</u>

The *Serial Transmit Object* receives an output data value from the DeviceNet master in the I/O Command Message. The object converts the output data into a serial message packet to transmit out the serial channel. The output data format can be a *Short_String* (byte array, with 1st byte = length), a signed or unsigned integer, or a real number data type. An integer or real number can be converted into ASCII characters that represent the numerical value before it is transmitted. *String1* and *String2* character strings can be placed before and/or after the converted data in the message packet, allowing you to build and transmit complex messages. The following examples show how the *Transmit Serial Object* can be configured to convert a data value into a message packet.

Instance Message Packet



1) *Output Data Bytes*

0x03 0x41 0x42 0x43

Short_String, data size = 4.
Use Data Field.
[0x41 0x42 0x43] = ‘ABC’

Transmitted Message Packet

‘ABC’

2) *Output Data Byte*

0x08

USINT integer, data size = 1,
width = 1, conversion = D.
Use Data Field.
[0x08] → ‘8’ (0x38)

Transmitted Message Packet

‘8’

3) *Output Data Bytes*

0xFF 0x47

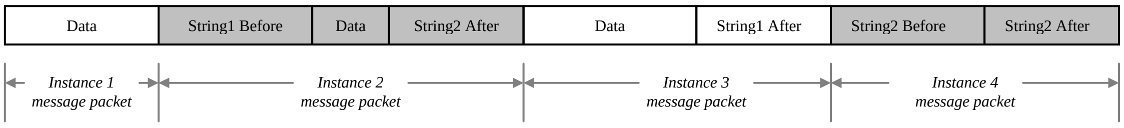
INT integer, data size = 2,
width = 4, conversion = D.
Use String1 Before [0x06, ‘VAL = ‘].
Use Data Field.
Use String2 After [0x02, ‘ F’].
[0xFF 0x47] = -185 → ‘-185’

Transmitted Message Packet

‘VAL = -185 F’

You can configure up to 8 *Serial Transmit Object* Instances, allowing you to transmit up to 8 different serial messages. You can also build complex messages by linking together each instance’s message packet. The *Serial Transmit Object* sequentially processes the I/O Command Message output data in order of instance number, starting with Instance 1. Each enabled instance converts its output data bytes into a message packet that is loaded into the transmit buffer. The following examples show how the *Serial Transmit Object* Instances create messages.

Transmitted Message


Data
String1 Before
Data
String2 After
Data
String1 After
String2 Before
String2 After
Instance 1 message packet
Instance 2 message packet
Instance 3 message packet
Instance 4 message packet

Example 1
Serial Transmit Object Instances 1 & 2 configured to transmit two different serial messages.

1)*Serial Transmit Object* Instance 1 configured to convert a Short_String data output value into message bytes.

DeviceNet output bytes = [0x04 0x30 0x31 0x32 0x33]
Transmitted message packet = ['0123']

2) *Serial Transmit Object* Instance 2 configured to convert an 8-bit unsigned integer (USINT) data output value into 3 ASCII characters. String1 'A EQUALS ' is placed before data, and String2 ' UNITS' is placed after data.

DeviceNet output byte = [0x10]
Transmitted message packet = ['A EQUALS 016 UNITS']

Example 2
Serial Transmit Object Instances 1, 2, 3 are configured to transmit a complex serial message using 3 output data values.

1)*Serial Transmit Object* Instance 1 Transmit Mode = Use String1 Before, Use Data, Use String2 After. String1 = [0x02, 'V ']. String2 = [0x01, ' =']. Data Type = USINT, width = 1, conversion = D with no leading zeros. Converts output data byte [0x08] into one ASCII char [0x38].

STX, 'V8 = '

2)*Serial Transmit Object* Instance 2 Transmit Mode = Use Data. Data Type = USINT, width = 3, conversion = D with leading zeros. Converts output data byte [0x22] into three ASCII chars [0x30 0x33 0x34].

'034'

3)*Serial Transmit Object* Instance 3 Transmit Mode = Use String1 Before, Use Data, Use String2 After. String1 = [0x04, ' REF ']. String2 = [0x01, 0x03]. Data Type = USINT, width = 2, conversion = D with leading zeros. Converts output data byte [0x13] into two ASCII chars [0x31 0x39].

' REF 19', ETX

The three message packets are transmitted sequentially, so complete message is:

STX, 'V8 = 034 REF 19' ETX

Asynchronous Serial Communictaion

Devices communicating on an asynchronous serial link exchange information one bit at a time. Each bit is transmitted for a specific period of time, defined by the baud rate. Devices use internal timing circuitry to measure the baud rate. There is no clocking signal between devices to synchronize the serial data flow, hence the term *asynchronous* serial communications.

Serial data bits are organized into bytes. When a data byte is asynchronously transmitted, it is preceded by a start bit, followed by the data bits, an optional parity bit, and one or more stop bits. There can be a variable transmission delay between successive serial data bytes, since each byte is framed by its own start and stop bits. The receiver starts saving serial bits after is receives a valid start bit (0), and stops when it receives the expected number of stop bits (1). The data byte's least-significant bit is transmitted first (data bit 0), and the most-signficant bit is transmitted last (data bit N).

[start bit] [data bit 0] [data bit 1] … [data bit N] [optional parity bit] [stop bit(s)]

The parity bit is used to detect single-bit errors in the transmission. The parity bit is automatically calculated and inserted by the transmitter. The receiver calculates the parity of an incoming byte, and compares it to the parity bit sent by the transmitter. If the two bit values do not match, then at least one serial bit value was corrupted during transmission.

Flow control allows the receiving device to regulate the rate of incoming data. Hardware flow control uses RTS/CTS signals between the devices to control the rate of transmission. Software flow control uses serial characters XON/OFF to control the rate. Flow control helps protect against lost data, if the receiving device cannot store incoming data fast enough, or if the receiving device's buffer is full and cannot accept more data until it processes existing data.

Data Conversion

The CDN36X gateway can either pass through received serial message bytes to your application, or pre-process an ASCII string into a numerical value. The gateway can transmit a string of message bytes sent by the application, or it can convert a numerical value into an ASCII string to be transmitted. Using the gateway's data conversion feature offloads this cumbersome task from your application program, especially if it is a PLC ladder-logic application. It also reduces the required number of DeviceNet input and output bytes, since converted values instead of entire message strings are transferred over DeviceNet.

The gateway conversion process supports the following data types:

Data Type		Data Size	Value Range
SINT	signed 8-bit integer	1	-128 to 127
INT	signed 16-bit integer	2	-32768 to 32767
USINT	unsigned 8-bit integer	1	0 to 255
UINT	unsigned 16-bit integer	2	0 to 65535
REAL	32-bit floating point	4	$\underline{+}$1.175E-38 to $\underline{+}$3.4028E+38
Short_String	(byte array)	2 to 128	string of bytes, 1st byte defines length

The *Serial Transmit Object* and *Serial Receive Object* Instances' attributes configure the conversion process for transmitted and received messages. The *Data Type* attribute selects the desired data type for an instance's *Receive Data* or *Transmit Data* value. The *Data Size* attribute represents the number of bytes used by the selected data type. You must select the maximum data size expected for your application if using the Short_String data type. The Short_String byte array format is a length byte followed by data bytes, so you must add one to the expected number of bytes. The table above defines the data size for all other data types.

The CDN36X gateway supports a maximum of 128 bytes for the 8 *Serial Transmit Object* Instances' *Transmit Data* values and 128 for the 8 *Serial Receive Object* Instances' *Receive Data* values, regardless of whether an instance is enabled or not. Configure all unused instances to USINT or SINT *Data Type*, because it has a 1 byte *Data Size*.

The *Width* attribute defines the number of ASCII bytes (1 to 16) used to represent a real or integer number. For received messages, *Width* defines the number of ASCII bytes that will be converted into a number. For transmitted messages, *Width* defines how many ASCII bytes will be generated to represent the number. The *Width* value must include ASCII sign (+/-), exponent (E), and decimal point (.) characters. *Width* is not used for Short_String data types.

The *Precision* attribute is only used for transmitted messages. It defines the number of digits (1 to 6) after the decimal point for a floating-point number. The gateway will automatically add trailing zeros to the converted number if needed. *Precision* is only used for the REAL data type.

The following examples show how to calculate the *Data Size*, *Width*, and *Precision* attributes for the different *Data Types*. Remember to add a length byte to the Short_String *Data Size*.

Data Type	ASCII chars	*Data Size*	*Width*	*Precision*
SINT	'-12'	1	3	not used
INT	'-12345'	2	6	not used
USINT	'123'	1	3	not used
UINT	'1234'	2	4	not used
REAL	'1.23E+4'	4	7	2
REAL	'-1.1234E-12'	4	11	4
Short_String	'ABCDEF'	7 (length=6)	not used	not used

The *Conversion* attribute is different for *Serial Receive Object* and *Serial Transmit Object*. For the *Serial Receive Object*, the *Conversion* attribute denotes if the ASCII bytes represent a decimal integer or a hexadecimal integer. If decimal is selected, then the gateway converts the ASCII bytes as a decimal number. If hexadecimal is selected, then the gateway converts the ASCII bytes as a hex number.

'1234' → If decimal, integer = 1234.
'1234' → If hexadecimal (0x1234), integer = 4660.

For the *Serial Transmit Object*, the *Conversion* attribute also denotes if the ASCII bytes represent a decimal or hexadecimal integer. If decimal is selected, then the gateway converts the integer into a decimal ASCII representation. If hexadecimal is selected, then the gateway converts the integer into a hex ASCII representation.

Integer = 1234 → If decimal, ASCII representation = '1234'
Integer = 1234 → If hexadecimal, ASCII representation = '04D2'

The *Serial Transmit Object Conversion* attribute can also be used to insert leading zeros into a converted number. If the ASCII representation of a number contains fewer characters than the selected *Width*, then leading zeros can added in front of the number.

Integer = 1234, Width = 7 → If leading zeros enabled, ASCII representation = '0001234'
Integer = 1234, Width = 7 → If leading zeros disabled, ASCII representation = '1234'

The following examples show a variety of different gateway data conversions for received and transmitted data values.

<u>Example 1 – Data Type = Short_String, Data Size = 9</u>
Received ASCII data is '12345678'. The *Serial Receive Object* Instance coverts this to 9 bytes of *Receive Data*, [0x08, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38] or [0x08, '12345678']. The first byte defines the *Short_String* length as 8 bytes.

<u>Example 2 – Data Type = Short_String, Data Size = 5</u>
Received ASCII data is '12345678'. The *Serial Receive Object* Instance converts this to 4 bytes of *Receive Data*, [0x04, 0x31, 0x32, 0x33, 0x34] or [0x04, '1234']. The first byte defines the *Short_String* length as 4 bytes. With *Data Size* = 5, only the first 4 data bytes are used.

<u>Example 3 – Data Type = Short_String, Data Size = 12</u>
Received ASCII data is 'ABCDEFGH'. The *Serial Receive Object* Instance converts this to 9 bytes of *Receive Data*, [0x08, 0x41, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48] or [0x08, 'ABCDEFGH']. The first byte defines the *Short_String* length as 8 bytes. Even though *Data Size* = 12, only the 8 received bytes are returned.

<u>Example 4 – Data Type = SINT, Width = 5, Conversion = Hex</u>
Received ASCII data is '18'. The *Serial Receive Object* Instance converts this to 1 byte of *Receive Data*, [0x18].

<u>Example 5 – Data Type = INT, Width = 4, Conversion = Decimal</u>
Received ASCII data is '-25'. The *Serial Receive Object* Instance converts this to 2 bytes of *Receive Data*, [0xFF 0xE7]. The ASCII '-25' decimal number converts to 0xFFE7.

<u>Example 6 – Data Type = REAL, Width = 13</u>
Received ASCII data is '-1.2345E-16'. The *Serial Receive Object* Instance converts this to 4 bytes of *Receive Data*, [0xNN, 0xNN, 0xNN, 0xNN]. This is the 32-bit floating-point representation for '–1.2345E-16'.

<u>Example 7 – Data Type = REAL, Width = 7</u>
Received ASCII data is '-1.2345E-16'. The *Serial Receive Object* Instance converts this to 4 bytes of *Receive Data*, [0xNN, 0xNN, 0xNN, 0xNN]. This is the 32-bit floating-point representation for '-1.2345'. With *Width* = 7, only the first 7 ASCII bytes are converted.

<u>Example 8 – Data Type = Short_String, Data Size = 8</u>
Transmit Data is [0x08, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38]. The *Serial Transmit Object* Instance converts this into 8 ASCII bytes '12345678'. Note the first *Transmit Data* byte defines the *Short_String* length in bytes. This length byte is not transmitted.

<u>Example 9 – Data Type = SINT, Width = 5, Conversion = Hex, Leading Zeros</u>
Transmit Data is [0x18]. The *Serial Transmit Object* Instance converts this into 5 ASCII bytes '00018'. Leading zeros are added to match the *Width* = 5.

<u>Example 10 – Data Type = INT, Width = 6, Conversion = Decimal, No Leading Zeros</u>
Transmit Data is [0xFF 0xE7]. The *Serial Transmit Object* Instance converts this into the ASCII string '-25'. The *Width* is 3 bytes greater than '-25', but leading zeros are not selected.

<u>Example 11 – Data Type = REAL, Width = 13, Precision = 6, Conversion = No Leading Zeros</u>
Transmit Data is [0xNN 0xNN 0xNN 0xNN], representing the real number –1.2345E-16. *The Serial Transmit Object* Instance converts this into the ASCII string '-1.234500E-16'. The *Precision* is 2 bytes greater than needed, so trailing zeros are added after the decimal point.

Serial Receive Example

The following example shows how the CDN36X gateway captures a serial message packet, processes the packet, converts the data into a number, and returns it as a DeviceNet input value.

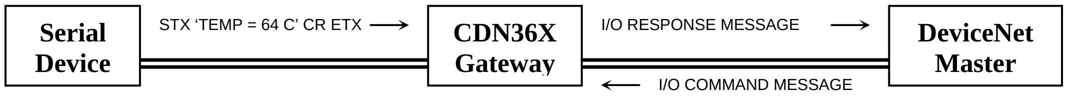
The gateway's *Serial Stream Object* is configured for the *List* delimiter mode, with a one-byte *Pre-Delimiter* string and a two-byte *Post-Delimiter* string. An incoming message must have matching delimiter strings to be accepted as a valid message packet.

The *Serial Receive Object* Instance 1 has the *Use Pre-String*, *Use Data*, and *Use Post-String* options selected for its *Receive Mode*. The message packet must have matching *Pre-String* and *Post-String* values before Instance 1 will process the data. The *Data Type* is configured for SINT, with a *Width* of 2 and *Conversion* set for decimal. The data field's two-byte ASCII string represents a decimal number, which Instance 1 converts into an 8-bit signed integer. The converted number is saved as the instance's new *Receive Data* value. The instance also toggles its *Receive Toggle* bit to signal the reception of new data.

When the gateway receives a DeviceNet I/O Command Message, it builds and sends a DeviceNet I/O Response Message. The *Transmit Acknowledge* bits from *Serial Transmit Object* instances 1-8 are loaded in the first byte. The *Receive Toggle* bits from *Serial Receive Object* instances 1-8 are loaded in the second byte. The *Receive Data* values from all enabled Serial Receive Object instances are loaded in the rest of the I/O Response data field. The I/O Response Message is sent to the DeviceNet master, where the data values are stored as DeviceNet inputs.

The user application receives the updated DeviceNet inputs. Program logic recognizes the state change in the *Serial Receive Object* Instance 1 *Receive Toggle* bit, indicating that Instance 1 has sent new data in the I/O Response Message. To acknowledge that it has read the new data, the user application toggles the Instance 1 *Receive Acknowledge* bit, which gets sent back to the gateway in the next I/O Command Message. When the *Serial Receive Object* Instance 1 gets the updated *Receive Acknowledge* bit, it can then process the next incoming serial message.

<u>Serial Receive Process</u>



1) Serial Device transmits message.
   Serial Data (ASCII) <STX> T E M P <SP> = <SP> **6** **4** <SP> C <CR> <ETX>
   Serial Data (hex) $\boxed{\text{0x02 0x54 0x45 0x4D 0x50 0x20 0x3D 0x20 \textbf{0x36 0x34} 0x20 0x43 0x0D 0x03}}$

2) *Serial Stream Object* receives message and loads into rx buffer without delimiters.
   Delimiter Mode = List
   Pre-Delimiter = [0x01 0x02]
   Post-Delimiter = [0x02 0x0D 0x03]
   $\boxed{\text{0x54 0x45 0x4D 0x50 0x20 0x3D 0x20 \textbf{0x36 0x34} 0x20 0x43}}$

3) *Serial Receive Object* Instance 1 processes message packet.
   Pre-String = [0x07, 'TEMP = ']
   Data = ['64']
   Post-String = [0x02, ' C']
   $\boxed{\text{0x54 0x45 0x4D 0x50 0x20 0x3D 0x20}}$ $\boxed{\textbf{0x36 0x34}}$ $\boxed{\text{0x20 0x43}}$

4) Instance 1 converts ASCII data to *Data Type* value.
   Data Type = SINT, Width = 2, Conversion = D
   0x36 0x34 ⇒ '64'⇒ 64 decimal ⇒ 0x40 hex
   $\boxed{\textbf{0x36 0x34}}$
   $\boxed{\textbf{0x40}}$

5) Instance 1 *Receive Data* value sent to DeviceNet master.
   I/O RESPONSE MESSAGE
   TA = Transmit Acknowledge bits
   RT = Receive Toggle bits (Instance 1 bit toggled 0-1)
   0x40 = Instance 1 Receive Data
   $\boxed{\text{TA}}$ $\boxed{\text{RT=xxxxxxx\textbf{1}}}$ $\boxed{\textbf{0x40}}$ $\boxed{\text{SRO3 Data}}$ $\boxed{\text{SRO6 Data}}$

6) Application acknowledges reading new *Receive Data*.
   I/O COMMAND MESSAGE
   TT = Transmit Toggle bits
   RA = Receive Acknowldege bits (Instance 1 bit set to 1)
   $\boxed{\text{TT}}$ $\boxed{\text{RA=xxxxxxx\textbf{1}}}$ $\boxed{\text{STO Data ...}}$

Serial Transmit Example

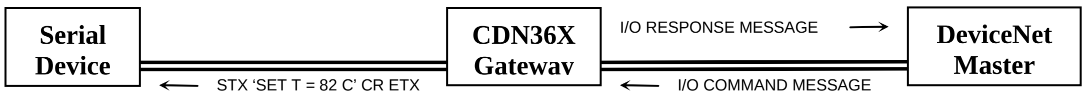
The following example shows how the CDN36X gateway receives DeviceNet output data, converts the data into an ASCII string, then builds and transmits a serial message using the converted string.

The gateway receives an I/O Command Message from the DeviceNet master. The first byte in the I/O Command data field contains the *Transmit Toggle* bits for *Serial Transmit Object* instances 1-8. The second byte contains the *Receive Acknowledge* bits for *Serial Receive Object* instances 1-8. The remaining data bytes are the *Transmit Data* for the enabled *Serial Transmit Object* instances. In this I/O Command, *Serial Transmit Object* Instance 1 receives a toggled *Transmit Toggle* bit, indicating the application has sent a new *Transmit Data* value to Instance 1.

Serial Transmit Object Instance 1 is configured for SINT *Data Type*, with *Width* of 2 and *Conversion* set for decimal with no leading zeros. Instance 1 converts its new *Transmit Data* value 0x52 into the ASCII string ‘82’. The Instance’s *Transmit Mode* attribute is set for *String1 Before, Data,* and *String2 After*. The resulting serial message that gets loaded into the gateway transmit buffer consists of [String1, ‘82’, String2]. The message will be sent when the serial channel transmitter is available.

When the new message is loaded in the transmit buffer, Instance 1 toggles its *Transmit Acknowledge* bit, indicating that is it ready to receive the next *Transmit Data* value. The updated *Transmit Acknowledge* bit is sent to the application in the next DeviceNet I/O Response Message. There may have been subsequent DeviceNet I/O message transactions in between the time Instance 1 received the new *Transmit Data* value and the time it toggles its *Transmit Acknowledge* bit.

Serial Transmit Process



1) *Transmit Data* received from DeviceNet master.
   I/O COMMAND MESSAGE TT=00000001 | RA | **0x52** | STO4 Data | STO7 Data
   TT = Transmit Toggle bits (Instance 1 bit toggled 0-1)
   RA = Receive Acknowledge bits
   0x52 = Instance 1 Transmit Data

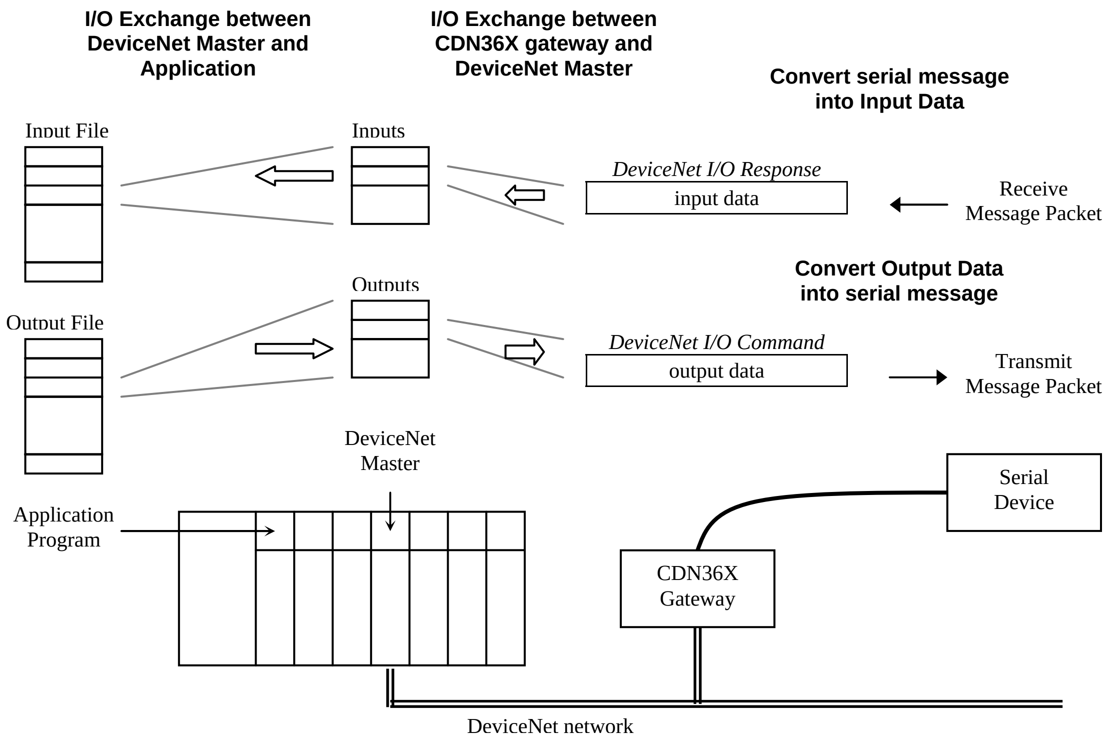
2) *Serial Transmit Object* Instance 1 converts data to ASCII.
   Data Type = SINT, Width = 2, Conversion = D **0x52**
   0x52 hex → 82 decimal → ‘82’ or [0x38 0x32] **0x38 0x32**

3) Instance 1 builds message & loads into transmit buffer.
   String1 = [0x09 0x02 ‘SET T = ‘] 0x02 0x53 0x45 0x54 0x20 0x54 0x20 0x3D 0x20 | 0x38 0x32 | 0x20 0x43 0x0D 0x03
   Data = [‘82’]
   String2 = [0x04 ‘ C’ 0x0D 0x03]

4) Gateway transmits message.
   Serial Data Stream (hex bytes) 0x02 0x53 0x45 0x54 0x20 0x54 0x20 0x3D 0x20 0x38 0x32 0x20 0x43 0x0D 0x03
   Serial Data Stream (ASCII) <STX> S E T <SP> T <SP> = <SP> 8 2 <SP> C <CR> <ETX>

Synchronization

There are four independent processes operating in a CDN36X gateway application. The first process is the exchange of input and output data between the user application program and the DeviceNet master. The second process is the exchange of input and output data between the gateway and DeviceNet master, using Polled or Change-of-State I/O messaging. The third process is receiving serial messages and converting it to input data. The fourth process is converting output data and transmitting it as serial messages. To ensure that no information is lost between the gateway's serial channel and the user application program, the CDN36X incorporates a receive synchronization feature and a transmit synchronization feature.



Receive Synchronization

The gateway receive synchronization feature is optional for each *Serial Receive Object* instance. It is enabled by the *Sync Enable* attribute. When enabled, the instance will not process a new message packet until the last *Receive Data* value has been read and acknowledged by the application program. When a *Serial Receive Object* Instance updates its *Receive Data* value, it also toggles its *Receive Toggle* bit to indicate a new data value is available. The user application monitors the *Receive Toggle* bit, and reads the Instance's *Receive Data* value when the bit changes state. Once the application has read and processed or stored the new *Receive Data* value, it acknowledges receipt by setting the Instance's *Receive Acknowledge* bit equal to the *Receive Toggle* bit. The Instance is now able to start processing the next serial message packet. The *Receive Toggle* and *Receive Acknowledge* bits are set to 0 at power-up.

The *Receive Acknowledge* bits are bit-mapped into the second byte of the DeviceNet I/O Command Message. *Serial Receive Object* Instance 1 *Receive Acknowledge* bit maps to bit 0, Instance 2 maps to bit 1, etc. The *Receive Toggle* bits are bit-mapped into the second byte of the DeviceNet I/O Response Message. *Serial Receive Object* Instance 1 *Receive Toggle* bit maps to bit 0, Instance 2 maps to bit 1, etc. This bit mapping makes it easy for ladder-logic applications to implement the gateway's receive-synchronization process. The following 2 ladder-logic rungs show how an application program can monitor for an instance's *Receive Toggle*, copy the new *Receive Data* value to save it, and set the *Receive Acknowledge* bit equal to the *Receive Toggle* bit. These rungs should be duplicated for each enabled *Serial Receive Object* instance.

```
|         bit = 1            bit = 0                                                          |
|------[ RX Toggle ]-------[ /RX Ack ]--------------------|------------( copy RX Data to variable )--------|
|                                                          |                                              |
|                                                          |--------------------( RX Ack )---------------------|
|                                                                        bit =1                           |
|         bit = 0            bit = 1                                                          |
|------[ /RX Toggle ]------[ RX Ack ]---------------------|------------( copy RX Data to variable )--------|
|                                                          |                                              |
|                                                          |--------------------( /RX Ack )--------------------|
|                                                                        bit =0                           |
```

Transmit Synchronization

The gateway transmit synchronization is always enabled for each *Serial Transmit Object* instance. An instance will not process its *Transmit Data* output bytes until its *Transmit Toggle* bit changes state. When the user application sends new *Transmit Data* to an instance, is must toggle that instance's *Transmit Toggle* bit. This enables the Instance to process the new output value and load the resulting serial message into the transmit buffer. When the message is loaded for transmission, the Instance acknowledges transmission by setting its *Transmit Acknowledge* bit equal to the *Transmit Toggle* bit. The application can now send the next *Transmit Data* value. The *Transmit Toggle* and *Transmit Acknowledge* bits are set to 0 at power-up.

The *Transmit Toggle* bits are bit-mapped into the first byte of the DeviceNet I/O Command Message. *Serial Transmit Object* Instance 1 *Transmit Toggle* bit maps to bit 0, Instance 2 maps to bit 1, etc. The *Transmit Acknowledge* bits are bit-mapped into the first byte of the DeviceNet I/O Response Message. *Serial Transmit Object* Instance 1 *Transmit Acknowledge* bit maps to bit 0, Instance 2 maps to bit 1, etc. This bit mapping makes it easy for ladder-logic applications to implement transmit synchronization. The following 2 ladder-logic rungs show how an application program writes an instance's *Transmit Data* value, toggles the *Transmit Toggle* bit, and waits for the *Transmit Acknowledge* bit to equal the *Transmit Toggle* bit before writing the next *Transmit Data* value. These rungs should be duplicated for each enabled *Serial Transmit Object* instance.

```
|           bit = 0              bit = 0                                                                       |
|------[ /TX Toggle ]-------[ /TX Ack  ]-------------------|-----------( copy variable to TX Data )---------|
|                                                          |                                               |
|                                                          |--------------------( TX Toggle  )-----------------|
|                                                                             bit =1                        |
|           bit = 1              bit = 1                                                                   |
|------[ TX Toggle ]-------[ TX Ack   ]--------------------|------------( copy variable to TX Data )--------|
|                                                          |                                               |
|                                                          |--------------------( /TX Toggle  )----------------|
|                                                                             bit =0                        |
```

Chapter 4 – Gateway Configuration

This chapter describes how to configure and operate the CDN36X gateway. You configure the gateway by reading and writing attribute values over its DeviceNet interface. There are a variety of DeviceNet configuration tools available. Simple configuration tools use GET_ATTRIBUTE and SET_ATTRIBUTE explicit message commands to read and write attribute values, addressing each attribute by its *Object*, *Instance*, and *Attribute* numbers. This information is contained in Chapter 5. More sophisticated configuration tools use EDS files to simplify attribute configuration. You can configure the gateway using pull-down menus, buttons, and data entry fields from the gateway's Electronic Data sheet (EDS) file. Chapter 6 contains a configuration example using the Rockwell Software RSNetworx™ program.

Configure DeviceNet Interface

Set the DeviceNet Baud Rate and MAC ID Address using the rotary switches. Configure switches before connecting to the DeviceNet network. There is either a small triangular indicator or white indicator on the switch. Use a small screwdriver to align that indicator with the desired setting. Remove the CDN36X cover if necessary to access the rotary switches.

DeviceNet Baud Rate Switch

Valid settings are 125K, 250K, 500K, or PGM. When PGM is selected, the CDN36X uses the baud rate saved in its retentive memory. To save a valid baud rate in memory, set the switch to the desired baud rate and power up the CDN36X for a few seconds. Power down and set the switch to PGM. You may also write to the DeviceNet Object Baud Rate attribute.

POSITION	SETTING	POSITION	SETTING
0	125 Kbps	5	invalid
1	250 Kbps	6	invalid
2	500 Kbps	7	invalid
3	invalid	8	invalid
4	invalid	9	PGM

MAC ID Switches

The two MAC ID switches represent decimal numbers from 00 to 99. The LSB switch selects the *Ones* digit and the MSB switch selects the *Tens* digit. Valid MAC IDs are 00 to 63. Setting a MAC ID address greater than 63 forces the gateway to use the MAC ID saved in retentive memory. To save a valid MAC ID in memory, set the switches to the desired MAC ID and power up the CDN36X for a few seconds. Power down and set the switches to value greater than 63. You may also write to the DeviceNet Object MAC ID attribute.

MSB	LSB	Address	MSB	LSB	Address
0	0 to 9	00 to 09	6	4 to 9	stored address
1	0 to 9	10 to 19	7	0 to 3	stored address
2	0 to 9	20 to 29	8	0 to 9	stored address
3	0 to 9	30 to 39	9	0 to 9	stored address
4	0 to 9	40 to 49			
5	0 to 9	50 to 59			
6	0 to 3	60 to 63			

Power Up Gateway

Connect the gateway to a DeviceNet network to power up the gateway.

DeviceNet Status LEDs

The CDN36X gateway has two bi-color status LEDs (*NET* and *MOD*) that indicate operational status. During power-up, the LEDs cycle through a sequence of alternating red and green. After power-up, the *NET* LED should be flashing green (or solid green if allocated to a DeviceNet master) and the *MOD* LED should be solid green. If this does not occur, disconnect from DeviceNet and verify all the switch settings. See Chapter 8 for additional troubleshooting topics.

State	**DeviceNet Status LED (*NET*)**
Off	No power.
Flashing Red	Configuration error. Check DeviceNet switch settings.
Solid Red	Unrecoverable error.
Flashing Green	Device not allocated to a DeviceNet master.
Solid Green	Normal runtime, device allocated as a slave.

State	**Module Status LED (*MOD*)**
Off	No power.
Flashing Red	Configuration error. Check object attribute settings.
Solid Red	Unrecoverable error.
Flashing Green	Not defined.
Solid Green	Normal Operation.

Serial Channel Status LEDs

The gateway has two bi-color LEDs to indicate serial channel activity. The *TX* LED flashes green when a packet is being transmitted. The *RX* LED flashes green when a packet is being received. A fault is indicated by solid red. After power-up, both LEDs should be off.

State	**Transmit Status LED (*TX*)**
Off	No data being transmitted
Flashing Red	Not defined
Solid Red	Transmit error (parity or overrun error)
Flashing Green	Data being transmitted
Solid Green	Not defined

State	**Receive Status LED (*RX*)**
Off	No data being received
Flashing Red	Not defined
Solid Red	Receive error (parity or overrun error)
Flashing Green	Data being received
Solid Green	Not defined

Register EDS File

If using a DeviceNet configuration tool that supports Electronic Data Sheet (EDS) files, you should now register the gateway’s EDS file with the software. The latest EDS file versions can be downloaded from *www.mksinst.com*. Select the EDS file that matches your gateway’s part number and firmware version. Follow your configuration tool instructions to register EDS file.

Configure Serial Channel

The *Serial Stream Obje*ct attributes control the physical layer settings for the CDN36X serial channel. These settings apply to all serial transmit and receive operations. The attributes also configure the reception of message packets. Before you can set or change any gateway configuration settings, make sure the gateway is not in the DeviceNet master scan list.

Serial Stream Object Instance Attributes (Class Code 64)

Number	Name	Data Type	Value
3	Baud Rate	UDINT	300, 1200, 2400, 4800, 9600, 19200 bps
4	Data Bits	USINT	7, 8
5	Parity	USINT	0 = no parity 1 = odd parity 2 = even parity 3 = mark 4 = space
6	Stop Bits	USINT	1, 2
7	Flow Control	USINT	0 = none 1 = XON / XOFF 2 = CTS / RTS
10	Delimiter Mode	USINT	Bit 0 – List mode Bit 1 – Timeout mode Bit 2 – Length mode
11	Pre-Delimiter List	Short_String	List mode – String of 1-9 bytes.
12	Post-Delimiter List	Short_String	List mode – String of 1-9 bytes.
13	Packet Timeout	USINT	Timeout mode – delay between received bytes (1-255 msec).
14	Packet Length	USINT	Length mode – Number of message bytes (1-128).
15	Serial Status	USINT	Bit 0 = RX buffer overrun error Bit 1 = RX parity error Bit 4 = TX buffer overrun error Bit 5 = TX parity error
16	Byte Swap	USINT	0 = disable. 1 = enable.
18	RS422 Mode	USINT	0 = 4-wire mode (RS422 full duplex) 1 = 2-wire mode (RS485 half duplex)
20	I/O Produce Size	UINT	Number of data bytes returned in an I/O Response Message.
21	I/O Consume Size	UINT	Number of data bytes expected in an I/O Command Message.

Baud Rate **–** Sets the serial channel’s data or baud rate. Enter *Baud Rate* in bits-per-second (bps) as a decimal number.

Data Bits **–** Selects the number of data bits in one serial byte. This number does not include start, parity, or stop bits.

Parity **–** Selects the parity type used in the serial byte. Selecting any parity option other than *NONE* adds 1 parity bit to the serial byte length.

Stop Bits **–** Selects the number of stop bits in one serial byte.

Flow Control **–** Selects the method of flow control used across the serial interface.

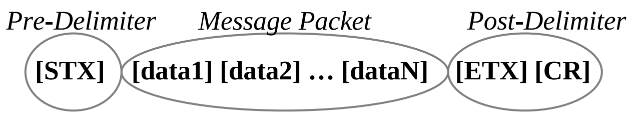
NONE means there is no flow control over the serial data exchange. The transmitting device can overflow the receiving device's buffer.

XON/XOFF is a software flow control option. Receiving device sends an XOFF character to the transmitting device when its buffer is full, stopping further transmission. It sends an XON character when it can again receive data. The XOFF and XON characters are not saved as message data.

CTS/RTS is an RS232 hardware flow control option, available only on the CDN366 gateway. The RTS is an output and CTS is an input signal. The gateway keeps RTS active (low) when it can receive data. It only transmits data when CTS is active (low).

Delimiter Mode **–** Defines how the gateway determines when it has received a message packet. The three delimiter modes are *List*, *Timeout*, and *Length*. Setting the appropriate bit in the *Delimiter Mode* byte selects the respective mode. The *Delimiter Mode* byte defines bits 0, 1, 2 only. Set the remaining bits 3 through 7 to zero.

List mode is used when a message packet is framed by a specific strings of *Pre-Delimiter* and *Post-Delimiter* bytes. The *Pre-Delimiter* signals the start of a new packet. The *Post-Delimiter* indicates the end of the packet. Each *Pre-Delimiter* and *Post-Delimiter* string can be from 1 to 9 bytes in length. When the gateway receives the *Pre-Delimiter* string, it saves the subsequent data bytes until the *Post-Delimiter* string is received. The *Pre-Delimiter* and *Post-Delimiter* bytes are not saved in the message packet. The following is a simple ASCII message example.



Length mode is used when every message packet contains the same number of bytes. The *Packet Length* attribute defines the packet size, from 1 to 128 bytes. The gateway saves serial bytes until it receives the specified number, and saves them as one message packet.

Timeout mode uses a delay between received data bytes to determine the end of a message packet. The *Packet Timeout* attribute defines the time-out period, from 1 to 255 milliseconds.

Pre-Delimiter List – Required for *List Mode*. Enter a string of 1 to 9 bytes that defines the start of a new serial message. Use Short_String data format, with 1st byte = string length. Example *Pre-Delimiter* is [0x01 0x02], where string length is 1 and delimiter character is 0x02 (STX). You must use the RSNetworx™ Class Instance Editor (Set Attribute Single command) to write a Short_String attribute value.

Post-Delimiter List **–** Required for *List Mode*. Enter a string of 1 to 9 bytes that defines the end of a serial message. Use Short_String data format, with 1st byte = string length. Example *Post-Delimiter* is [0x02 0x0D 0x03], where string length is 2 and delimiter characters are 0x0D (CR) and 0x03 (ETX). You must use the RSNetworx™ Class Instance Editor (Set Attribute Single command) to write a Short_String attribute value.

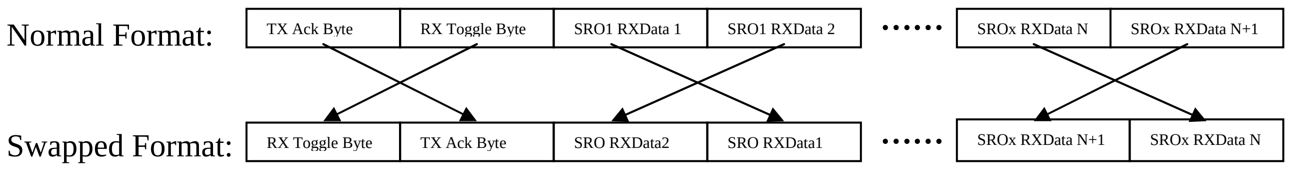
Packet Timeout **–** Required for *Timeout Mode*. Defines the timeout period between received bytes that indicates the end of a message packet (1-255 milliseconds).

Packet Length **–** Required for *Length Mode*. Defines the message packet size (1-128 bytes).

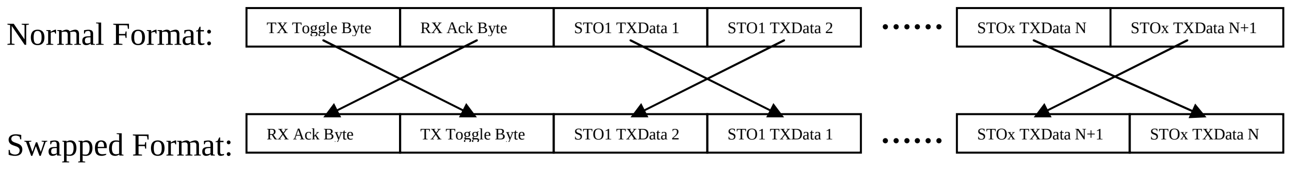
Serial Status **–** Serial transmitter and receiver error bits. To clear an error, you must either reset the gateway or clear the error bit using a SET_ATTRIBUTE explicit message command.

Byte Swap **–** Defines if CDN36X gateway swaps I/O message bytes. When disabled, the CDN36X I/O messages are mapped into the PLC DeviceNet Scanner module memory in low-byte / high byte format. Many PLC processors use word-aligned data tables, so working with ASCII strings in a low byte / high byte format is difficult. By enabling the CDN36X byte swap feature, the gateway automatically swaps each pair of contiguous bytes in the I/O messages. This allows you to send output message data in high byte / low byte format, and receive input message data in high byte / low byte format.

The following example shows how the CDN36X byte-swap feature modifies the input bytes in a DeviceNet I/O Response Message.



The following example shows how the CDN36X byte-swap feature interprets the output bytes in a DeviceNet I/O Command Message.



RS422 Mode **–** CDN367 only. Selects between RS422 4-wire or RS485 2-wire operation.

I/O Produce Size **–** Read-only attribute is the number of input bytes sent in an I/O Response Message. Gateway calculates based on enabled *Serial Receive Object* Instances' *Data Size*.

I/O Consume Size **–** Read-only attribute is the number of output bytes expected in an I/O Command Message. Gateway calculates based on enabled *Serial Transmit Object* Instances' *Data Size*.

Serial Receive Object Settings

There are eight identical *Serial Receive Object* instance attribute sets that can be configured in the CDN36X gateway. This section describes how to configure a single *Serial Receive Object* instance. Repeat this step for each desired instance.

Serial Receive Object Instance Attributes (Class Code 65)

Number	Name	Data Type	Value
3	Receive Data	Data Type	Received message data. Returned in I/O Response Message.
4	Receive Toggle	BOOL	Gateway toggles (0-1, 1-0) to indicate new Receive Data value.
5	Receive Acknowledge	BOOL	When Sync Enabled, user application must set this bit to match Receive Toggle before next message is processed.
6	Receive Mode	USINT	Bit 0 – use Data Field Bit 1 – use Pre-String Field Bit 2 – use Post-String Field
7	Pre-String	Short_String	String of 1-9 bytes.
8	Post-String	Short_String	String of 1-9 bytes.
9	Data Type	USINT	194 (0xC2) = SINT (1 byte) 195 (0xC3) = INT (2 bytes) 198 (0xC6) = USINT (1 byte) 199 (0xC7) = UINT (2 bytes) 202 (0xCA) = REAL (4 bytes) 218 (0xDA) = Short String (Data Size bytes)
10	Data Size	USINT	1-128
11	Width	USINT	1-16
13	Conversion	USINT	‘D’ (0x44) = decimal integer. ‘X’ (0x58) = hexadecimal integer.
14	Pad Char	CHAR	Pad byte value. Pad Poll Response if Rx data does not fill up Poll response message data.
15	Data in I/O Response	BOOL	0 = no, 1 = yes
16	Enabled	BOOL	0 = disabled, 1 = enabled
17	Sync Enabled	BOOL	0 = disabled, 1 = enabled

Receive Data – Data from the last valid message packet. Returned in I/O Response Message.

Receive Toggle – Toggles (0 to 1, or 1 to 0) when a message packet has been received, processed, and saved as *Receive Data*. Indicates new input data in the I/O Response Message.

Receive Acknowledge – When *Sync Enabled* is set, User Application must set this bit equal to *Receive Toggle* after it reads the Instance’s *Receive Data* from the I/O Response Message. The Instance will not process the next message packet until the *Receive Acknowledge* bit equals the *Receive Toggle* bit.

Receive Mode – Defines how the Instance processes message packet bytes. The Instance can search for 3 fields – *Pre-String*, *Data*, and *Post-String*. Set associated bits (0, 1, 2) to use the desired fields. Set the remaining bits 3 through 7 to zero.

[Pre-String] [Data] [Post-String]

Pre-String attribute defines the byte string for the *Pre-String* field. *Data Size* attribute defines the number of bytes expected in the *Data* field. *Post-String* attribute defines the byte string for the *Post-String* field.

When *Use Pre-String* bit is set, the Instance searches the message packet bytes for a match to the stored *Pre-String*. If a match is not found, the message packet is ignored.

When *Use Data* bit is set, the Instance converts the *Data Size* number of message bytes into a value defined by *Data Type*, and saves it as *Receive Data*. If there are not enough message bytes, the message packet is ignored. You must select *Use Data* in order to send input data to the DeviceNet master.

When *Use Post-String* bit is set, the Instance searches the message packet bytes for a match to the stored *Post-String*. If a match is not found, the message packet is ignored.

***Pre-String* –** Required if *Use Pre-String* selected in *Receive Mode*. Enter a string of 1-9 bytes in Short_String data format, with 1st byte = string length. Example *Pre-String* is [0x01 0x41], where string length is 1 and pre-string character is 0x41 (‘A’). You must use the RSNetworx™ Class Instance Editor (Set Attribute Single command) to write a Short_String attribute value.

***Post-String* –** Required if *Use Post-String* selected in *Receive Mode*. Enter a string of 1-9 bytes in Short_String data format, with 1st byte = string length. Example *Post-String* is [0x02 0x42 0x43], where string length is 2 and post-string characters are 0x42 (‘B’) and 0x43 (‘C’). You must use the RSNetworx™ Class Instance Editor (Set Attribute Single command) to write a Short_String attribute value.

***Data Type* –** The Instance converts received ASCII message data into the selected data type for *Receive Data*.

Decimal	Hex.	Data Type	Date Size (byte)	Value Range
194	0xC2	SINT (signed 8-bit integer)	1	-128 ~ 127
195	0xC3	INT (signed 16-bit integer)	2	-32768 ~ 32767
198	0xC6	USINT (unsigned 8-bit integer)	1	0 ~ 255
199	0xC7	UINT (unsigned 16-bit integer)	2	0 ~ 65535
202	0xCA	REAL (32-bit floating point value)	4	±1.175E-38 ~ ±3.4028E+38
218	0xDA	Short_String	Set by data size attribute, Max. 240 bytes	

Enter number from decimal (or hex) column to select the desired data type. *Data Size* column defines the number of data bytes for *Receive Data*. For *Short_String*, set *Data Size* attribute to the desired number of data bytes (plus 1 length byte). Set the *Width* attribute to the expected number of ASCII bytes to be converted into a real or integer number.

***Data Size* –** Required for Short_String *Data Type*. Defines the maximum number of bytes in a Short-String, plus one length byte (2-128). The first byte in a Short_String defines the string length.

The *Data Size* attributes for all 8 *Serial Receive Object* instances must sum to a total less than or equal to 128 bytes, regardless of whether an instance is enabled or not. Set unused instances' *Data Type* attributes to USINT or SINT, which have 1-byte *Data Size*.

Width **–** Required for SINT, INT, USINT, UINT, REAL *Data Types*. Defines the number of ASCII bytes (1-16) to be converted into an integer or real number.

Conversion **–** Required for SINT, INT, USINT, UINT *Data Types*. Denotes if the ASCII bytes represent a decimal integer ('D' or 0x44) or a hexadecimal integer ('X' or 0x58).

Pad Char **–** Byte value to pad the RX Message bytes.

Data in I/O Response **–** Enables the Instance to send its *Receive Data* value as input data in the I/O Response Message. Typically you would always have this attribute enabled when the Instance is enabled. For more complex applications, the Instance's *Receive Data* can be read using the Get_Attribute command (explicit message), reducing the number of input bytes in the I/O Response Message. All disabled Instances must have this attribute set to zero.

Enabled **–** Enables the *Serial Receive Object* instance. When enabled, the Instance processes received messages based on its *Receive Mode* and *Data Type*. This attribute must be disabled for all unused *Serial Receive Object* Instances.

Sync Enabled **–** Enables receive synchronization with the user application. When enabled, the instance will not update its *Receive Data* until the *Receive Acknowledge* bit matches the *Receive Toggle* bit. Enabling receive synchronization ensures that the user application does not miss any received message data between polls.

Serial Transmit Object Settings

There are eight identical *Serial Transmit Object* instance attribute sets that can be configured in the CDN36X gateway. This section describes how to configure a single *Serial Transmit Object* instance. Repeat this step for each desired instance.

Serial Transmit Object Instance Attributes (Class Code 66)

Number	Name	Data Type	Value
3	Transmit Data	Data Type	Message data to transmit. Received in I/O Command Message.
4	Transmit Toggle	BOOL	User app toggles (0-1, 1-0) to indicate new Transmit Data value.
5	Transmit Acknowledge	BOOL	Gateway sets this bit to match Transmit Toggle when the latest Transmit Data message has been sent.
6	Transmit Mode	USINT	Bit 0 – use Data Bit 1 – use String1 before data Bit 2 – use String2 before data Bit 3 – use String1 after data Bit 4 – use String2 after data
7	String1	Short_String	String of 1-9 bytes.
8	String2	Short_String	String of 1-9 bytes.
9	Data Type	USINT	194 (0xC2) = SINT (1 byte) 195 (0xC3) = INT (2 bytes) 198 (0xC6) = USINT (1 byte) 199 (0xC7) = UINT (2 bytes) 202 (0xCA) = REAL (4 bytes) 218 (0xDA) = Short String (Data Size bytes)
10	Data Size	USINT	1-128
11	Width	USINT	1-16
12	Precision	USINT	0-6
13	Conversion	USINT	Bit 0 – hex (0 for decimal, 1 for hex) Bit 7 – use leading zeros to pad number
15	Data In I/O Command	BOOL	0 = no, 1 = yes

Transmit Data **–** Data to be transmitted by this Instance as a serial message packet. The *Transmit Data* value is typically received in the DeviceNet I/O Command Message.

Transmit Toggle **–** User application must toggle this bit when it sends a new *Transmit Data* value in the I/O Command Message. Instance will only process and transmit a *Transmit Data* value once after this bit is toggled.

Transmit Acknowledge – Gateways sets equal to *Transmit Toggle* when the current *Transmit Data* message packet is queued in transmit buffer, indicating the Instance is ready new data.

Transmit Mode **–** Defines the message packet structure to be transmitted. The message packet can consist of 5 fields – *String1 Before*, *String2 Before*, *Data*, *String1 After*, and *String2 After*. Set associated bits (0, 1, 2, 3, 4) to enable the desired fields. Set the remaining bits 5-7 to zero.

[String1 Before] [String2 Before] [Data] [String1 After] [String2 After]

String1 attribute defines the byte string for the *String1 Before* and *String1 After* fields. *Data Size* attribute defines the number of bytes expected in the *Data* field. *String2* attribute defines the byte string for the *String2 Before* and *String2 After* fields. While the *Data* field is typically selected, you can configure an Instance to transmit a predefined message using String1 and/or String2, without requiring any output data bytes from the DeviceNet master.

When the *String1 Before* and/or *String2 Before* bits are set, the Instance places the respective byte string(s) at the beginning of the message packet. If both options are selected, String1 is placed before String2. The Instance then loads the converted data bytes in the Data field. If the *String1 After* and/or *String2 After* bits are set, the Instance places the respective byte string(s) at the end of the message packet. If both options are selected, then String1 is placed before String2. The message packet is then loaded into the Transmit Buffer to be sent out the serial channel.

***String1* –** Required if *String1 Before* or *String1 After* selected in *Transmit Mode*. Enter a string of 1-9 bytes in Short_String data format, with 1st byte = string length. Example *String1* is [0x02 0x41 0x42], where string length is 2 and post-string characters are 0x41 (‘A’) and 0x42 (‘B’). You must use the RSNetworx™ Class Instance Editor (Set Attribute Single command) to write a Short_String attribute value.

***String2* –** Required if *String2 Before* or *String2 After* selected in *Transmit Mode*. Enter a string of 1-9 bytes in Short_String data format, with 1st byte = string length. Example *String2* is [0x03 0x43 0x44 0x45], where string length is 3 and post-string characters are 0x43 (‘C’), 0x44 (‘D’), and 0x45 (‘E’). You must use the RSNetworx™ Class Instance Editor (Set Attribute Single command) to write a Short_String attribute value.

***Data Type* –** Defines the *Transmit Data* attribute data type.

Decimal	Hex.	Data Type	Date Size (byte)	Value Range
194	0xC2	SINT (signed 8-bit integer)	1	-128 ~ 127
195	0xC3	INT (signed 16-bit integer)	2	-32768 ~ 32767
198	0xC6	USINT (unsigned 8-bit integer)	1	0 ~ 255
199	0xC7	UINT (unsigned 16-bit integer)	2	0 ~ 65535
202	0xCA	REAL (32-bit floating point value)	4	±1.175E-38 ~ ±3.4028E+38
218	0xDA	Short_String	Set by data size attribute, Max. 240 bytes	

Enter number from decimal (or hex) column to select the desired data type. *Data Size* column defines the number of data bytes for *Transmit Data*. For *Short_String,* set *Data Size* attribute to the desired number of data bytes (plus 1 length byte). Set the *Width* attribute to the expected number of ASCII bytes to be converted into a real or integer number. Set the *Precision* attribute to the desired number of digits after the decimal point in a real number.

***Data Size* –** Required for Short_String *Data Type*. Defines the maximum number of bytes in a Short-String, plus one length byte (2-128). The first byte in a Short_String defines the string length

The *Data Size* attributes for all 8 *Serial Receive Object* instances must sum to a total of less than or equal to 128 bytes, and all 8 *Serial Transmit Object* instances must sum to a total less than or equal to 128 bytes, regardless of whether an instance is enabled or not. Set unused instances' *Data Type* attributes to USINT or SINT, which have a 1-byte *Data Size*.

***Width* –** Required for SINT, INT, USINT, UINT, REAL *Data Types*. Defines the number of ASCII bytes (1-16) that will represent the integer or real number.

***Precision* –** Required for REAL *Data Type*. Defines the number of digits (0 to 6) after the real number decimal point. Gateway adds trailing zeros to the converted value if needed.

***Conversion* –** Selects *Leading Zeros* and *Hex or Decimal* representation. The following are valid options for the *Conversion* byte.

bit 7	bit 0	decimal	hex	description
0	0	0	0x00	no leading zeros, decimal integer
0	1	1	0x01	no leading zeros, hexadecimal integer
1	0	128	0x80	leading zeros, decimal integer
1	1	129	0x81	leading zeros, hexadecimal integer

When bit 0 = 0, the ASCII bytes represent the *Transmit Data* integer number in a decimal format. When bit 0 = 1, the ASCII bytes represent the integer number in a hexadecimal format. This bit only applies to SINT, INT, USINT, and UINT Data Types.

When bit 7 = 1, leading zeros are added to real and integer numbers as needed to match the *Width* setting. This bit only applies to SINT, INT, USINT, UINT, and REAL Data Types.

***Data in I/O Command* –** Enables the Instance to use output data from the I/O Command Message as its *Transmit Data* value. This bit must be set to enable the *Serial Transmit Object* Instance. All unused instances must be disabled.

Configure DeviceNet Master Scanlist

You must calculate the number of input and output bytes required by your CDN36X configuration before you can add the gateway to the DeviceNet master scan list. You need to configure the DeviceNet master to send the specific number of output bytes in its I/O Command Message, and receive the specific number of input bytes in the gateway's I/O Response Message. Once the input and output bytes are mapped in the DeviceNet master, the user application program will be able to read and write data values to the input and output bytes.

I/O Consume Size

The *I/O Consume Size* is the size (in bytes) of the I/O Command Message data field that is sent by the DeviceNet master to the CDN36X.

I/O Command data:
[TX Toggle bits 1-8][RX Ack bits 1-8][TX Data Instance 1] … [TX Data Instance 8]

The first byte contains the *Transmit Toggle* bits for all 8 *Serial Transmit Object* Instances. The second byte contains the *Receive Acknowledge* bits for all 8 *Serial Receive Object* Instances. These two bytes are used by the CDN36X and application program to synchronize the transmit and receive operations. The remaining bytes are the *Transmit Data* attributes for every enabled *Serial Transmit Object* Instance. The number of bytes is determined by the *Data Size* configured for each enabled Instance. For real and integer numbers, the *Data Size* is predefined by the selected *Data Type*. For Short_String data type, the *Data Size* attribute is user-defined.

The following equation is used to calculate the CDN36X *I/O Consume Size*. Only include the *Data Size* for enabled *Serial Transmit Object* Instances.

	Transmit Toggle Byte	1
	Receive Acknowledge Byte	1
	Serial Transmit Object Instance 1 *Data Size*	____
	Serial Transmit Object Instance 2 *Data Size*	____
	Serial Transmit Object Instance 3 *Data Size*	____
	Serial Transmit Object Instance 4 *Data Size*	____
	Serial Transmit Object Instance 5 *Data Size*	____
	Serial Transmit Object Instance 6 *Data Size*	____
	Serial Transmit Object Instance 7 *Data Size*	____
+	*Serial Transmit Object* Instance 8 *Data Size*	____
	I/O Consume Size	____

Once you have the gateway configured for your application, you can also read the *Serial Stream Object's I/O Consume Size* attribute to find out the required number of input bytes.

I/O Produce Size

The *I/O Produce Size* is the size (in bytes) of the I/O Response Message data field that is sent from the CDN36X to the DeviceNet master.

I/O Response data:
[TX Ack bits 1-8][RX Toggle bits 1-8][RX Data Instance 1] … [RX Data Instance 8]

The first byte contains the *Transmit Acknowledge* bits for all 8 *Serial Transmit Object* Instances. The second byte contains the *Receive Toggle* bits for all 8 *Serial Receive Object* Instances. These two bytes are used by the CDN36X and application program to synchronize the transmit and receive operations. The remaining bytes are the *Receive Data* attributes for every enabled *Serial Receive Object* Instance. The number of bytes is determined by the *Data Size* configured for each enabled Instance. For real and integer numbers, the *Data Size* is predefined by the selected *Data Type*. For Short-String data byte, *the Data Size* attribute is user-defined.

The following equation is used to calculate the CDN36X I/O Produce Size. Only include the *Data Size* for enabled *Serial Receive Object* Instances.

	Transmit Acknowledge Byte	1
	Receive Toggle Byte	1
	Serial Receive Object Instance 1 *Data Size*	____
	Serial Receive Object Instance 2 *Data Size*	____
	Serial Receive Object Instance 3 *Data Size*	____
	Serial Receive Object Instance 4 *Data Size*	____
	Serial Receive Object Instance 5 *Data Size*	____
	Serial Receive Object Instance 6 *Data Size*	____
	Serial Receive Object Instance 7 *Data Size*	____
+	*Serial Receive Object* Instance 8 *Data Size*	____
	I/O Produce Size	____

Once you have the gateway configured for your application, you can also read the *Serial Stream Object's I/O Produce Size* attribute to find out the required number of output bytes.

Chapter 5 – DeviceNet Specifications

This chapter describes the CDN36X gateway DeviceNet specifications.

DeviceNet Message Types

The CDN36X is a Group 2 Slave Device that supports the following message types.

CAN IDENTIFIER	**GROUP 2 MESSAGE TYPE**
10xxxxxx111	Duplicate MAC ID Check Message
10xxxxxx110	Unconnected Explicit Request Message
10xxxxxx101	Master I/O Command Message
10xxxxxx100	Master Explicit Request Message

xxxxxx = CDN36X MAC ID

DeviceNet Class Services

The CDN36X is a Group 2 Slave Device that supports the following class services and instance services.

SERVICE CODE	**SERVICE NAME**
05 (0x05)	Reset
14 (0x0E)	Get Attribute Single
16 (0x10)	Set Attribute Single
75 (0x4B)	Allocate Group 2 Identifier Set
76 (0x4C)	Release Group 2 Identifier Set

DeviceNet Object Classes

The CDN366 device supports the following DeviceNet object classes.

CLASS CODE	**OBJECT TYPE**
01 (0x01)	Identity
02 (0x02)	Router
03 (0x03)	DeviceNet
04 (0x04)	Assembly
05 (0x05)	Connection
64 (0x40)	Serial Stream Object
65 (0x41)	Serial Receive Object
66 (0x42)	Serial Transmit Object

IDENTITY OBJECT

The Identity Object is required on all DeviceNet devices. It provides product identification of and general information.

Identity Object				**Class Code 01 (0x01)**
Class Attribute	**Access**	**Name**	**Type**	**Value**
1	Get	Revision	UINT	1
2	Get	Max Object Instance	UINT	1
6	Get	Max Class Identifier	UINT	7
7	Get	Max Instance Attribute	UINT	7
Instance Attribute	**Access**	**Name**	**Type**	**Value**
1	Get	Vendor	UINT	59 = D.I.P. Products
2	Get	Product Type	UINT	12 = Communications
3	Get	Product Code	UINT	5856
4	Get	Revision	STRUCT of	
		Major Revision	USINT	1
		Minor Revision	USINT	9
5	Get	Device Status	WORD	bit 0 = owned (0 available, 1 allocated) bit 2 = configured (0 no, 1 yes) bit 4-7 = vendor specific (0) bit 8 = minor configuration fault bit 9 = minor device fault bit 10 = major configuration fault bit 11 = major device fault bit 1, 3, 12-15 = reserved (0)
6	Get	Serial Number	UDINT	unique serial number for every device
7	Get	Product Name	STRUCT of	
		Length	USINT	6
		Name	STRING [6]	CDN3xx
8	Get	State	USINT	0 = nonexistent 1 = device self-testing 2 = standby 3 = operational 4 = major recoverable fault 5 = major unrecoverable fault

Common Services

Service Code	**Class**	**Instance**	**Service Name**
05 (0x05)	No	Yes	Reset
14 (0x0E)	Yes	Yes	Get_Attribute_Single

ROUTER OBJECT

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 CDN36X device.

Router Object				**Class Code 02 (0x02)**
Class Attribute	**Access**	**Name**	**Type**	**Value**
1	Get	Revision	UINT	1
6	Get	Max Class Identifier	UINT	7
7	Get	Max Instance Attribute	UINT	2
Instance Attribute	**Access**	**Name**	**Type**	**Value**
2	Get	Number of Connections	UINT	2

Common Services

Service Code	**Class**	**Instance**	**Service Name**
14 (0x0E)	Yes	Yes	Get_Attribute_Single

DEVICENET OBJECT

The DeviceNet Object contains information about the CDN36X DeviceNet interface configuration.

DeviceNet Object				**Class Code 03 (0x03)**
Class Attribute	**Access**	**Name**	**Type**	**Value**
1	Get	Revision	UINT	2
Instance Attribute	**Access**	**Name**	**Type**	**Value**
1	Get/Set	MAC ID	USINT	Settable only if MAC ID switches > 63. Valid numbers are 0 to 63. Returns last value or switch value.
2	Get/Set	Baud Rate	USINT	Settable only if Baud switch = PGM. Valid settings are 0=125K, 1=250K, 2= 500K. Returns last value or switch value.
3	Get/Set	Bus Off Interrupt	BOOL	0 = hold CAN in OFF state (default) 1 = reset CAN
4	Get/Set	Bus Off Counter	USINT	Writing this attribute forces counter value to zero.
5	Get	Allocation Information	STRUCT of	
		Choice Byte	BYTE	bit 0 = explicit msg, set to 1 to allocate bit 1 = polled IO, set to 1 to allocate bit 2 = strobed IO, not supported bits 3-7 = reserved, set to 0
		Master Node Address	USINT	Allocated to this DeviceNet master
6	Get	MAC ID Switch Changed	BOOL	0 = No Change. 1 = Changed since last Power-up or Reset.
7	Get	Baud Rate Switch Changed	BOOL	0 = No Change. 1 = Changed since last Power-up or Reset.
8	Get	MAC ID Switch Value	USINT	Physical switch setting, 00 to 99.
9	Get	Baud Rate Switch Value	USINT	Physical switch setting, 0 to 9.

Common Services

Service Code	**Class**	**Instance**	**Service Name**
14 (0x0E)	Yes	Yes	Get_Attribute_Single
16 (0x10)	No	Yes	Set_Attribute_Single
75 (0x4B)	No	Yes	Allocate Master/Slave
76 (0x4C)	No	Yes	Release Master/Slave

ASSEMBLY OBJECT

The Assembly Object instances bind attributes of multiple objects to allow data to or from each object to be sent or received over a single connection.

Assembly Object				**Class Code 04 (0x04)**
Class Attribute	**Access**	**Name**	**Type**	**Value**
1	Get	Revision	UINT	2
2	Get	Max Class ID	UINT	2
Instance Attribute	**Access**	**Name**	**Type**	**Value**
3	Get	Data Stream	note 1	Instance 100 for input data stream. Instance 101 for output data stream.

Common Services

Service Code	**Class**	**Instance**	**Service Name**
14 (0x0E)	Yes	Yes	Get_Attribute_Single
16 (0x10)	No	Yes	Set_Attribute_Single

Instance 100 Input Data Stream and Instance 101 Output Data Stream are structured as either an array of bytes or as a Short_String consisting of a single byte length field and N data bytes. The Input Data Stream is the data returned in the I/O Response Message. The Output Data Stream is the data returned in the I/O Command Message. See Chapter 3 for a complete description of the I/O Format.

I/O Response:
[TX Ack bits 1-8][RX Toggle bits 1-8][RX Data Instance 1][RX Data Instance 2] … [RX Data Instance 8]

I/O Command:
[TX Toggle bits 1-8][RX Ack bits 1-8][TX Data Instance 1][TX Data Instance 2] … [TX Data Instance 8]

CONNECTION OBJECT

The Connection Object instances manage the characteristics of each communication connection. The CDN36X is a Group 2 Only Slave device that supports 1 Explicit Message Connection and 1 I/O Message Connection.

Connection Object				**Class Code 05 (0x05)**
Class Attribute	**Access**	**Name**	**Type**	**Value**
1	Get	Revision	UINT	1
Instance Attribute	**Access**	**Name**	**Type**	**Value**
1	Get	State	USINT	0 = non-existent 1 = configuring 2 = established 3 = timed out
2	Get	Instance Type	USINT	0 = Explicit Message 1 = I/O Message
3	Get	Transport Class Trigger	USINT	0x83 for Explicit Message 0x82 for I/O Message
4	Get	Production Connection	UINT	Explicit Message: 10xxxxxx011 = produced connection id 10xxxxxx100 = consumed connection id I/O Message: 01111xxxxxx = produced connection id 10xxxxxx101 = consumed connection id
5	Get	Consumed Connection	UINT	
6	Get	Initial Communication Characteristics	USINT	0x21 for Explicit Message 0x01 for I/O Message
7	Get	Production Size	UINT	67 for Explicit Message See Stream Object for I/O Message
8	Get	Consumed Size	UINT	71 for Explicit Message See Stream Object for I/O Message
9	Get/Set	Expected Packet Rate	UINT	Default 2500 msec
12	Get/Set	Timeout Action	USINT	0 = Timeout (Explicit Message default) 1 = Auto Delete 2 = Auto Reset (I/O Message default)
13	Get	Production Path Length	USINT	0 for Explicit Message 6 for I/O Message
14	Get	Production Path	STRUCT of	Null for Explicit Message STRUCT for I/O Message
		Log. Seg., Class	USINT	0x20
		Class Number	USINT	0x04
		Log. Seg., Instance	USINT	0x24
		Instance Number	USINT	0x01
		Log. Seg., Attribute	USINT	0x30
		Attribute Number	USINT	0x03
15	Get	Consumed Path Length	USINT	0 for Explicit Message 6 for I/O Message

16	Get	Consumed Path	STRUCT of	Null for Explicit Message STRUCT for I/O Message
		Log. Seg., Class	USINT	0x20
		Class Number	USINT	0x04
		Log. Seg., Instance	USINT	0x24
		Instance Number	USINT	0x02
		Log. Seg., Attribute	USINT	0x30
		Attribute Number	USINT	0x03
17	Get	Production Inhibit	UINT	0

Common Services

Service Code	**Class**	**Instance**	**Service Name**
05 (0x05)	Yes	Yes	Reset
14 (0x0E)	Yes	Yes	Get_Attribute_Single
16 (0x10)	No	Yes	Set_Attribute_Single

SERIAL STREAM OBJECT

The Serial Stream Object configures the CDN36X serial channel.

Serial Stream Object				**Class Code 64 (0x40)**
Class Attribute	**Access**	**Name**	**Type**	**Value**
1	Get	Revision	UINT	1
2	Get	Max Object Instance	UINT	1
6	Get	Max Class Identifier	UINT	7
7	Get	Max Instance Attribute	UINT	14
Instance Attribute	**Access**	**Name**	**Type**	**Value**
3	Get/Set	Baud Rate	UDINT	300, 1200, 2400, 4800, 9600, 19200 bits per sec.
4	Get/Set	Data Bits	USINT	7, 8
5	Get/Set	Parity	USINT	0 = no parity 1 = odd parity 2 = even parity 3 = mark 4 = space
6	Get/Set	Stop Bits	USINT	1, 2
7	Get/Set	Flow Control	USINT	0 = none 1 = XON / XOFF 2 = CTS / RTS
10	Get/Set	Delimiter Mode	USINT	Bit 0 – List mode Bit 1 – Timeout mode Bit 2 – Length mode
11	Get/Set	Pre-Delimiter List	Short_String	List mode – String of 1-9 bytes.
12	Get/Set	Post-Delimiter List	Short_String	List mode – String of 1-9 bytes.
13	Get/Set	Packet Timeout	USINT	Timeout mode – delay between received bytes (1-255 msec).
14	Get/Set	Packet Length	USINT	Length mode – Number of message bytes (1-128).
15	Get/Set	Serial Status	USINT	Bit 0 = RX buffer overrun error Bit 1 = RX parity error Bit 4 = TX buffer overrun error Bit 5 = TX parity error
16	Get/Set	Byte Swap	USINT	0 = disable 1 = enable
18	Get/Set	RS422 Mode	USINT	0 = 4-wire mode (RS422 full duplex) 1 = 2-wire mode (RS485 half duplex)
20	Get/Set	I/O Produce Size	UINT	Number of data bytes returned in a I/O Response Message.
21	Get/Set	I/O Consume Size	UINT	Number of data bytes expected in a I/O Command Message.

Common Services

Service Code	**Class**	**Instance**	**Service Name**
05 (0x05)	No	Yes	Reset
14 (0x0E)	Yes	Yes	Get_Attribute_Single
16 (0x10)	No	Yes	Set_Attribute_Single

SERIAL RECEIVE OBJECT

The Serial Receive Object instances receive and process serial messages, and send the converted data to DeviceNet master in the I/O Response Message.

Serial Receive Object				**Class Code 65 (0x41)**
Class Attribute	**Access**	**Name**	**Type**	**Value**
1	Get	Revision	UINT	1
2	Get	Max Object Instance	UINT	8
6	Get	Max Class Identifier	UINT	7
7	Get	Max Instance Attribute	UINT	17
Instance Attribute	**Access**	**Name**	**Type**	**Value**
3	Get/Set	Receive Data	Data Type	Received message data. Returned in I/O Response Message.
4	Get	Receive Toggle	BOOL	Gateway toggles (0-1, 1-0) to indicate new Receive Data value.
5	Get/Set	Receive Acknowledge	BOOL	When Sync Enabled, user application must set this bit to match Receive Toggle before next message is processed.
6	Get/Set	Receive Mode	USINT	Bit 0 – use Data Field Bit 1 – use Pre-String Field Bit 2 – use Post-String Field
7	Get/Set	Pre-String	Short_String	String of 1-9 bytes.
8	Get/Set	Post-String	Short_String	String of 1-9 bytes.
9	Get/Set	Data Type	USINT	194 (0xC2) = SINT (1 byte) 195 (0xC3) = INT (2 bytes) 198 (0xC6) = USINT (1 byte) 199 (0xC7) = UINT (2 bytes) 202 (0xCA) = REAL (4 bytes) 218 (0xDA) = Short String (Data Size bytes)
10	Get/Set	Data Size	USINT	1-128
11	Get/Set	Width	USINT	1-16
13	Get/Set	Conversion	USINT	‘D’ (0x44) = ASCII represents decimal integer. ‘X’ (0x58) = ASCII represents hex integer.
14	Get/Set	Pad Char	CHAR	Pad byte value. Pad Poll Response if Rx data does not fill up Poll response message data.
15	Get/Set	Data in I/O Response	BOOL	0 = no, 1 = yes
16	Get/Set	Enabled	BOOL	0 = disabled, 1 = enabled
17	Get/Set	Sync Enabled	BOOL	0 = disabled, 1 = enabled

Common Services

Service Code	**Class**	**Instance**	**Service Name**
05 (0x05)	No	Yes	Reset
14 (0x0E)	Yes	Yes	Get_Attribute_Single
16 (0x10)	No	Yes	Set_Attribute_Single

SERIAL TRANSMIT OBJECT

The Serial Transmit Object instances receive data from DeviceNet master in the I/O Command Message, convert it and transmit the resulting message out the serial channel.

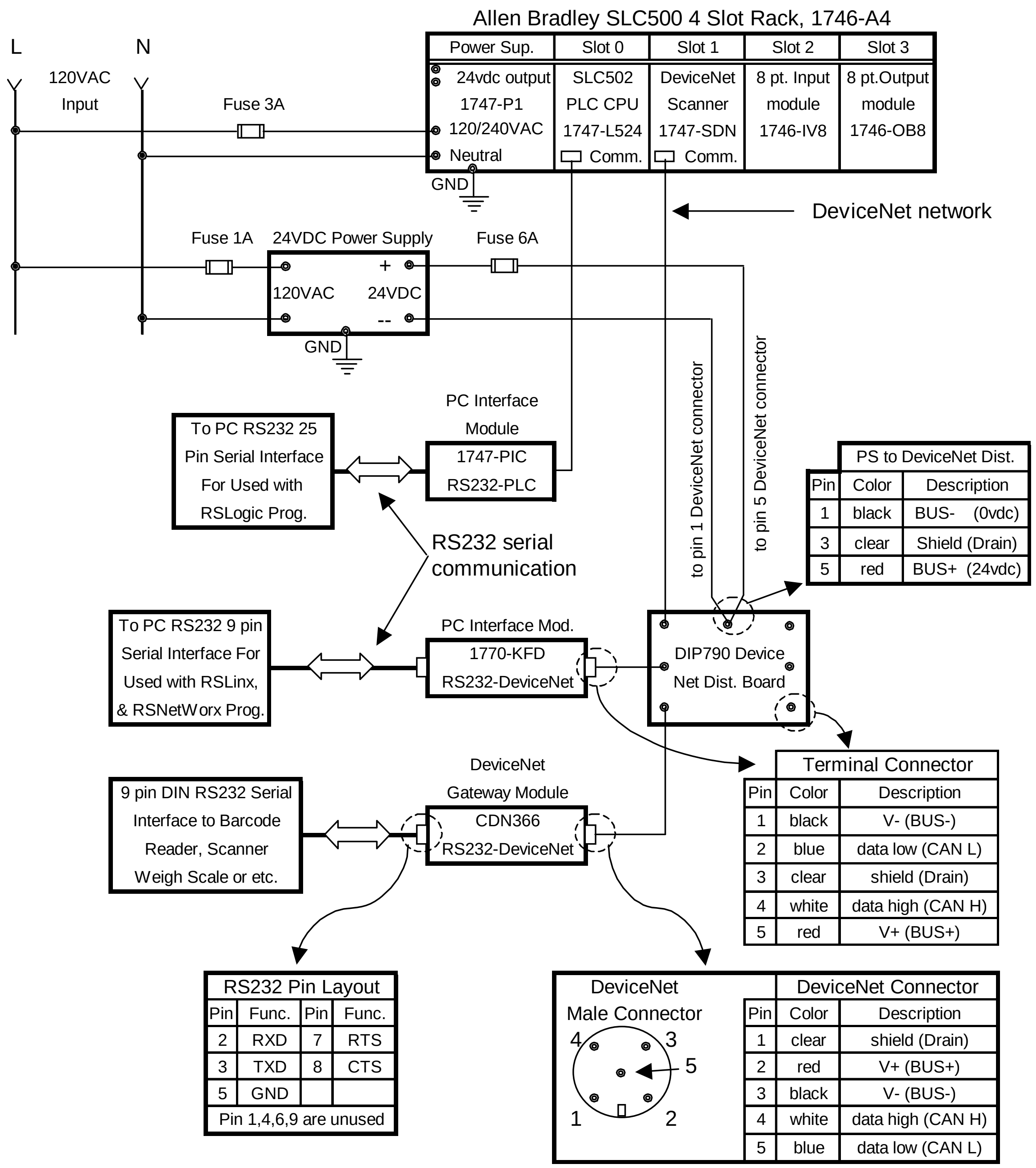
Serial Transmit Object				**Class Code 66 (0x42)**
Class Attribute	**Access**	**Name**	**Type**	**Value**
1	Get	Revision	UINT	1
2	Get	Max Object Instance	UINT	8
6	Get	Max Class Identifier	UINT	7
7	Get	Max Instance Attribute	UINT	15
Instance Attribute	**Access**	**Name**	**Type**	**Value**
3	Get/Set	Transmit Data	Data Type	Message data to transmit. Received in I/O Command Message.
4	Get/Set	Transmit Toggle	BOOL	User app toggles (0-1, 1-0) to indicate new Transmit Data value.
5	Get	Transmit Acknowledge	BOOL	Gateway sets this bit to match Transmit Toggle when the latest Transmit Data message has been sent.
6	Get/Set	Transmit Mode	USINT	Bit 0 – use Data Bit 1 – use String1 before data Bit 2 – use String2 before data Bit 3 – use String1 after data Bit 4 – use String2 after data
7	Get/Set	String1	Short_String	String of 1-9 bytes.
8	Get/Set	String2	Short_String	String of 1-9 bytes.
9	Get/Set	Data Type	USINT	194 (0xC2) = SINT (1 byte) 195 (0xC3) = INT (2 bytes) 198 (0xC6) = USINT (1 byte) 199 (0xC7) = UINT (2 bytes) 202 (0xCA) = REAL (4 bytes) 218 (0xDA) = Short String (Data Size bytes)
10	Get/Set	Data Size	USINT	1-128
11	Get/Set	Width	USINT	1-16
12	Get/Set	Precision	USINT	0-6
13	Get/Set	Conversion	USINT	Bit 0 – hex (0 for decimal, 1 for hex) Bit 7 – use leading zeros to pad number
15	Get/Set	Data In I/O Command	BOOL	0 = no, 1 = yes

Common Services

Service Code	**Class**	**Instance**	**Service Name**
05 (0x05)	No	Yes	Reset
14 (0x0E)	Yes	Yes	Get_Attribute_Single
16 (0x10)	No	Yes	Set_Attribute_Single

Chapter 6 – RSNetworx™ Configuration Example

This chapter shows how to set up configure a CDN366 gateway using the Rockwell Software RSNetworx™ software and your gateway's Electronic Data Sheet (EDS) file. The system configuration uses an Allen-Bradley 1770-KFD DeviceNet adapter (MAC ID 62) to connect the PC running RSNetworx™ to the DeviceNet network. A SLC500 system with a 1747-SDN DeviceNet Scanner (MAC ID 00) is the DeviceNet master. CDN366 gateway has MAC ID 03.



PS to DeviceNet Dist.

Pin	Color	Description
1	black	BUS- (0vdc)
3	clear	Shield (Drain)
5	red	BUS+ (24vdc)

Terminal Connector

Pin	Color	Description
1	black	V- (BUS-)
2	blue	data low (CAN L)
3	clear	shield (Drain)
4	white	data high (CAN H)
5	red	V+ (BUS+)

RS232 Pin Layout

Pin	Func.	Pin	Func.
2	RXD	7	RTS
3	TXD	8	CTS
5	GND		

Pin 1,4,6,9 are unused

DeviceNet Connector

Pin	Color	Description
1	clear	shield (Drain)
2	red	V+ (BUS+)
3	black	V- (BUS-)
4	white	data high (CAN H)
5	blue	data low (CAN L)

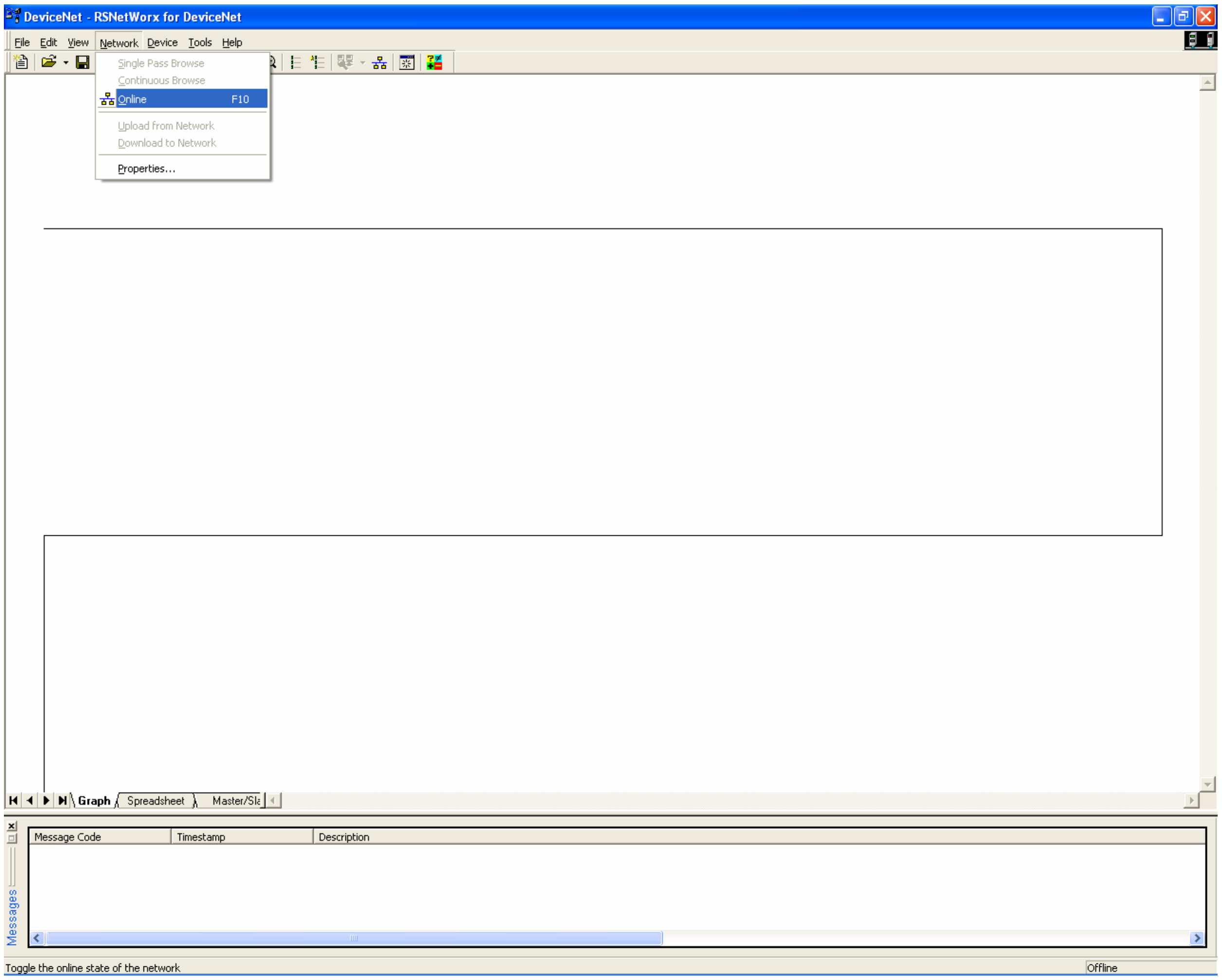
Figure 1. CDN366 Integrate with Allen Bradley SLC500

Configure DeviceNet Interface

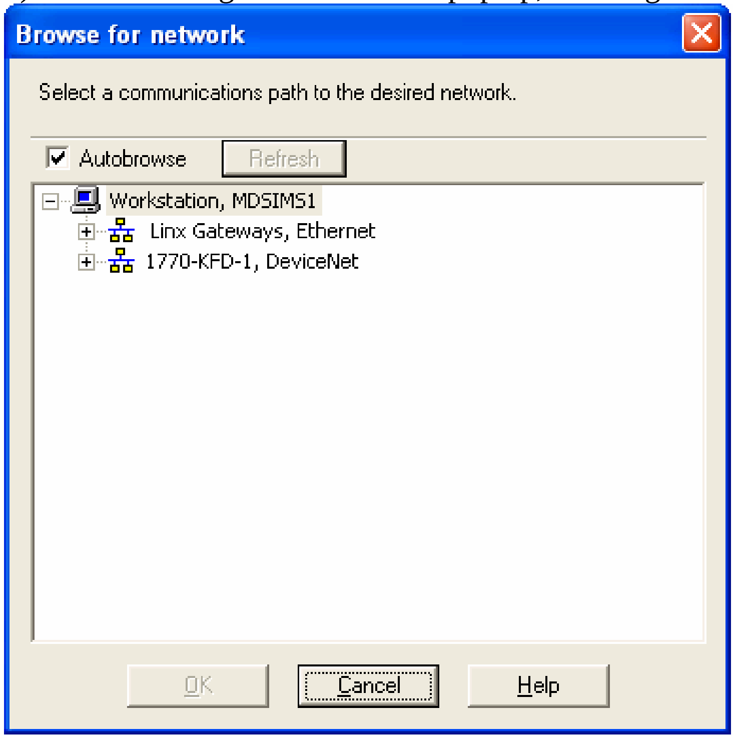
Follow instructions in Chapter 4 to set the gateway's rotary switches to 125Kbps baud rate and MAC ID to 03. Connect the gateway to the DeviceNet network to power it up. During power-up, the *NET* and *MOD* LEDs cycle through a sequence of alternating red and green. After power-up, the *NET* LED should be flashing green and the *MOD* LED should be solid green.

Connect & Register EDS File

1) Start up the RSNetworx program. Select the *Online* operation from the *Network* menu.



2) The following text box should pop up, showing the networks connected to your computer.


Browse for network

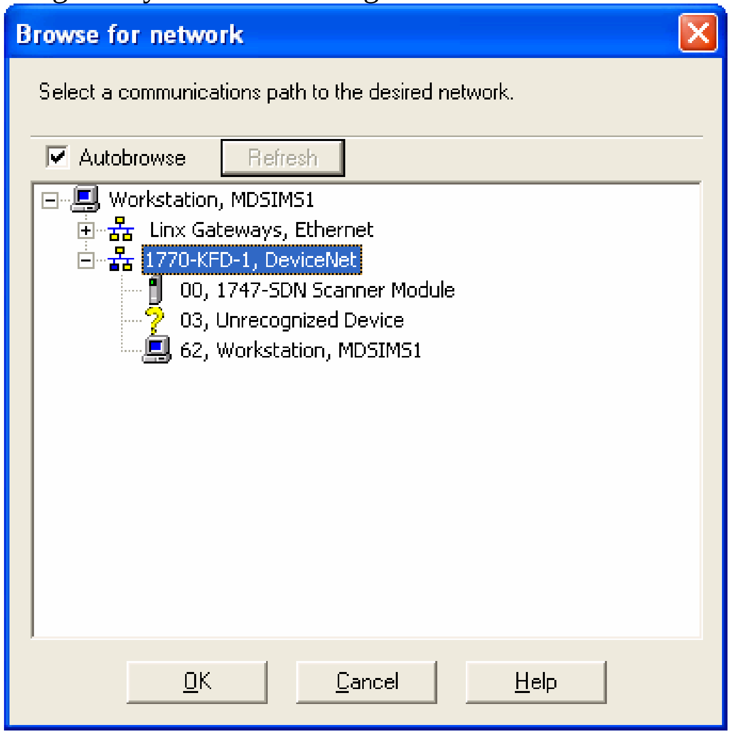
Select a communications path to the desired network.

☑ Autobrowse Refresh

- Workstation, MDSIMS1
  - Linx Gateways, Ethernet
  - 1770-KFD-1, DeviceNet

OK Cancel Help

3) Click on the *1770-KFD-1* + to show all connected DeviceNet devices. The gateway is at MAC ID 03, verifying its DeviceNet connection. It is an *Unrecognized Device* until the gateway's EDS file is registered with RSNetworx.


Browse for network

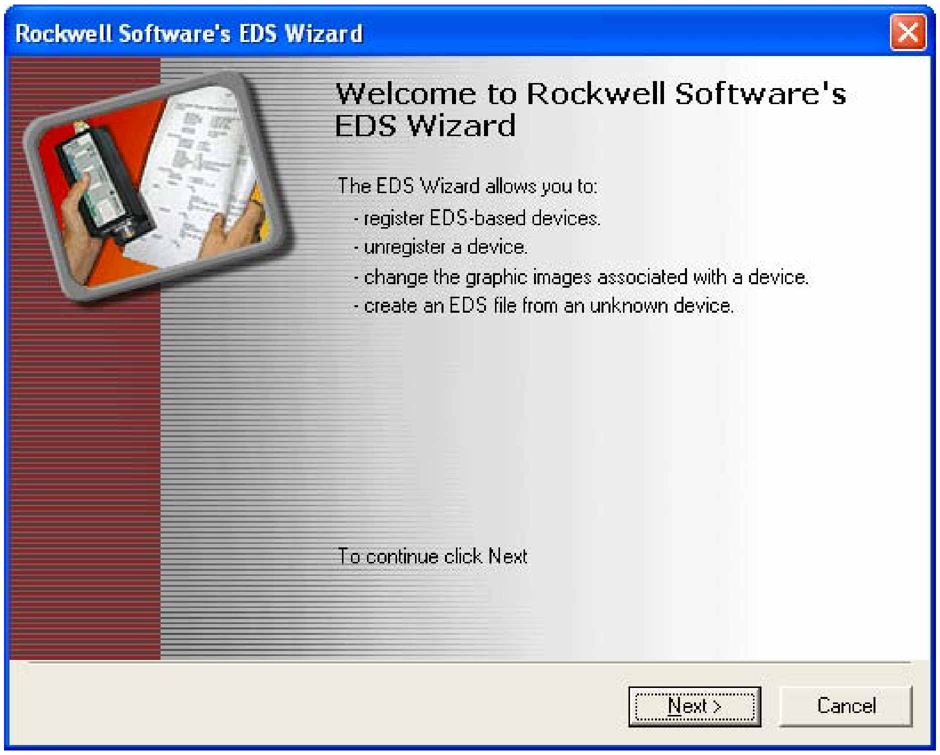
Select a communications path to the desired network.

☑ Autobrowse Refresh

- Workstation, MDSIMS1
  - Linx Gateways, Ethernet
  - 1770-KFD-1, DeviceNet
    - 00, 1747-SDN Scanner Module
    - 03, Unrecognized Device
    - 62, Workstation, MDSIMS1

OK Cancel Help

4) Click *Cancel* to close *Browse for network* window. Select the *EDS Wizard…* operation from the *Tools* menu. Click *Next>* to continue.



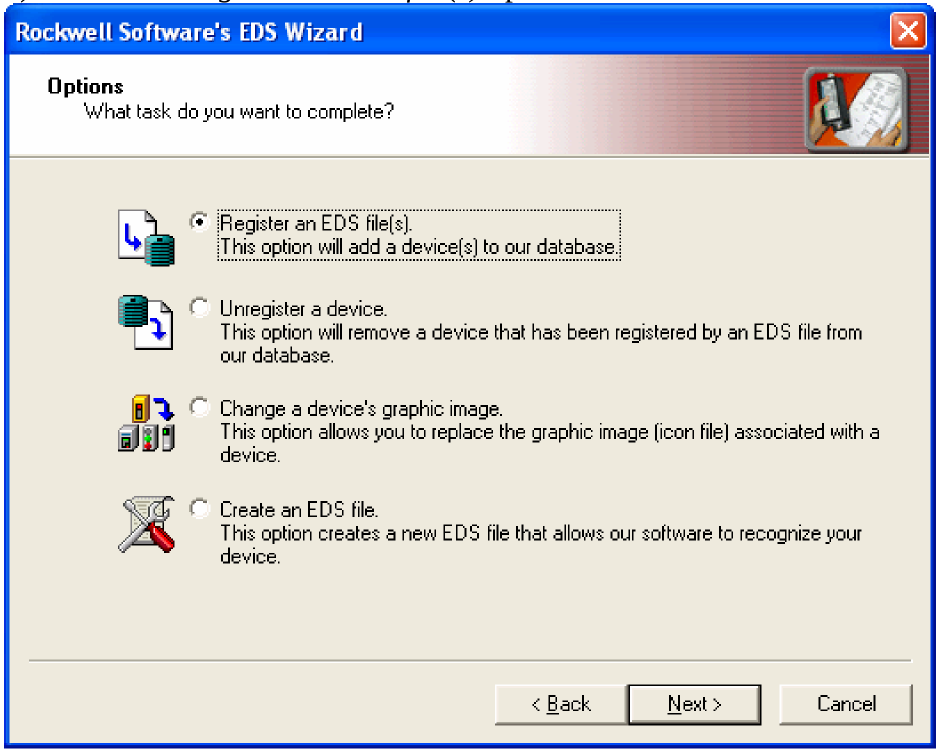
Rockwell Software's EDS Wizard

Welcome to Rockwell Software's EDS Wizard

The EDS Wizard allows you to:
- register EDS-based devices.
- unregister a device.
- change the graphic images associated with a device.
- create an EDS file from an unknown device.

To continue click Next

5) Select the *Register an EDS file(s)* option and click *Next>*.



Rockwell Software's EDS Wizard

Options
What task do you want to complete?

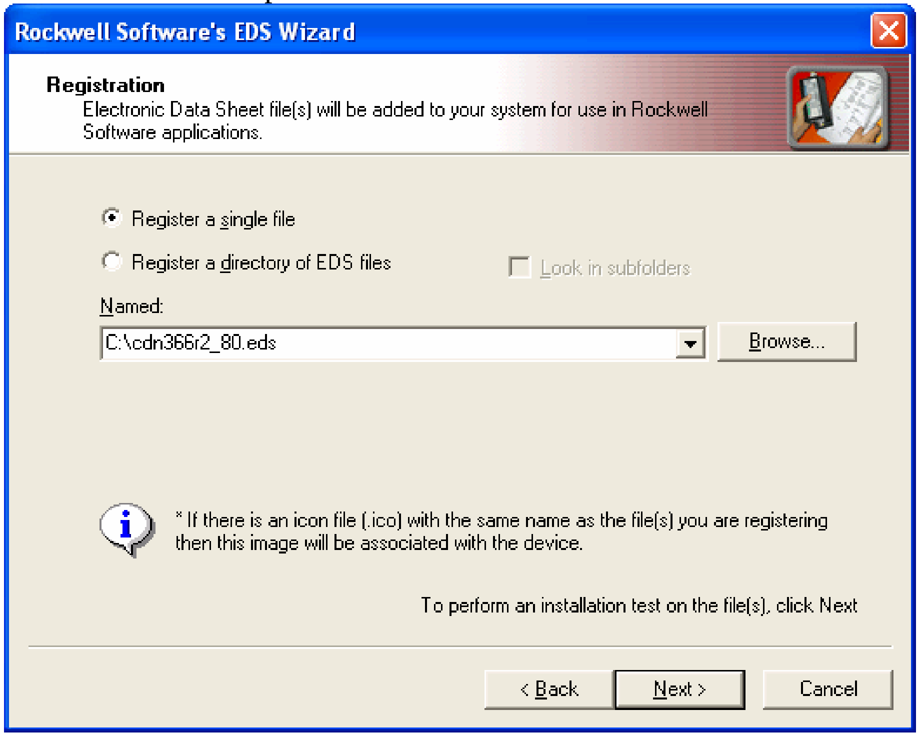
[✓ Register an EDS file(s).]
This option will add a device(s) to our database.

Unregister a device.
This option will remove a device that has been registered by an EDS file from our database.

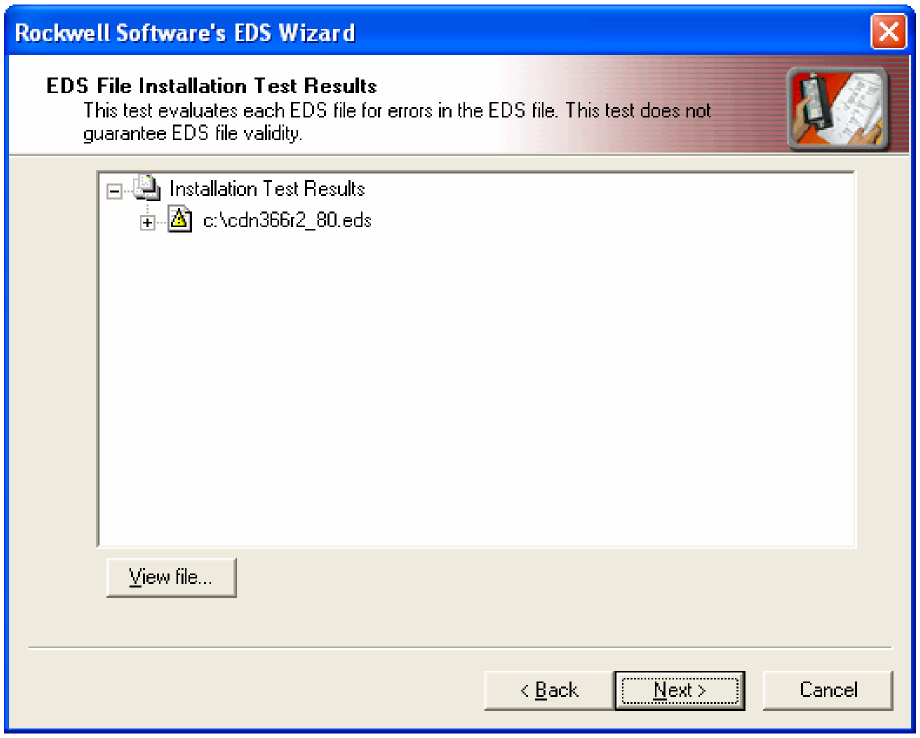
Change a device's graphic image.
This option allows you to replace the graphic image (icon file) associated with a device.

Create an EDS file.
This option creates a new EDS file that allows our software to recognize your device.

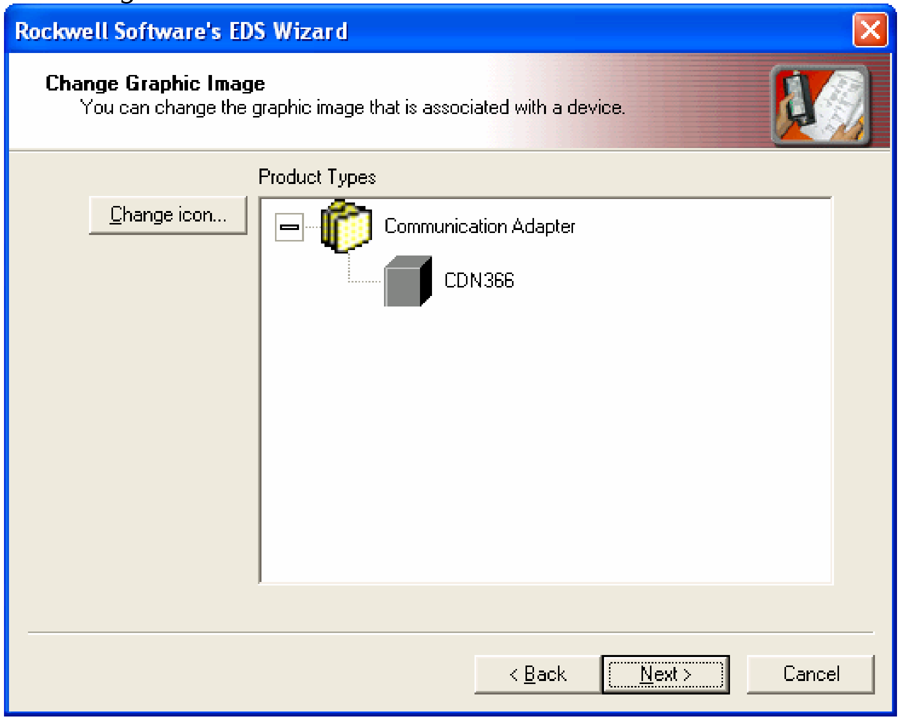
6) Select *Register a single file* option. *Browse* for your gateway's EDS file. You can download the latest EDS and ICON files from the www.mksinst.com website. Click *Next>* when you have the correct path and EDS file name in the *Named:* box.


Rockwell Software's EDS Wizard
Registration
Electronic Data Sheet file(s) will be added to your system for use in Rockwell Software applications.
Register a single file
Register a directory of EDS files
Look in subfolders
Named:
C:\cdn366r2_80.eds
Browse...
* If there is an icon file (.ico) with the same name as the file(s) you are registering then this image will be associated with the device.
To perform an installation test on the file(s), click Next
< Back
Next >
Cancel

7) The next screen shows the RSNetworx installation test results. Click *View file…* to view the actual EDS file text. Click *Next>* to continue.


Rockwell Software's EDS Wizard
EDS File Installation Test Results
This test evaluates each EDS file for errors in the EDS file. This test does not guarantee EDS file validity.
Installation Test Results
c:\cdn366r2_80.eds
View file...
< Back
Next >
Cancel

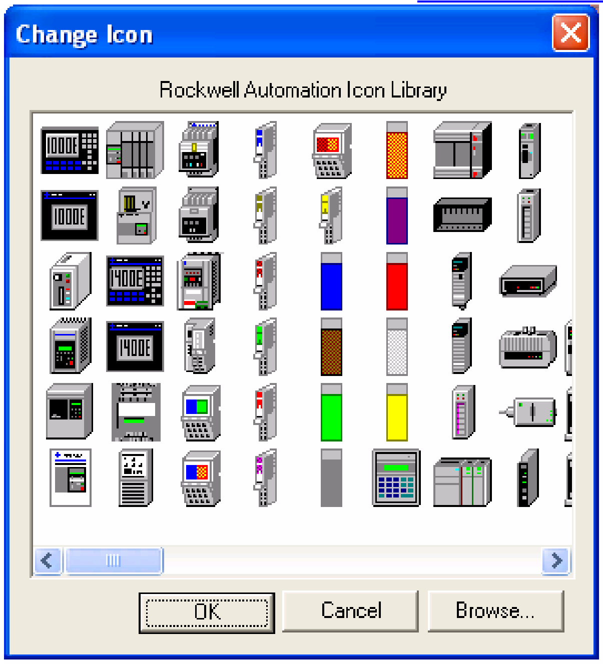
8) The next screen allows you to customize the gateway's icon for RSNetworx. Click on *Change icon…*


Rockwell Software's EDS Wizard

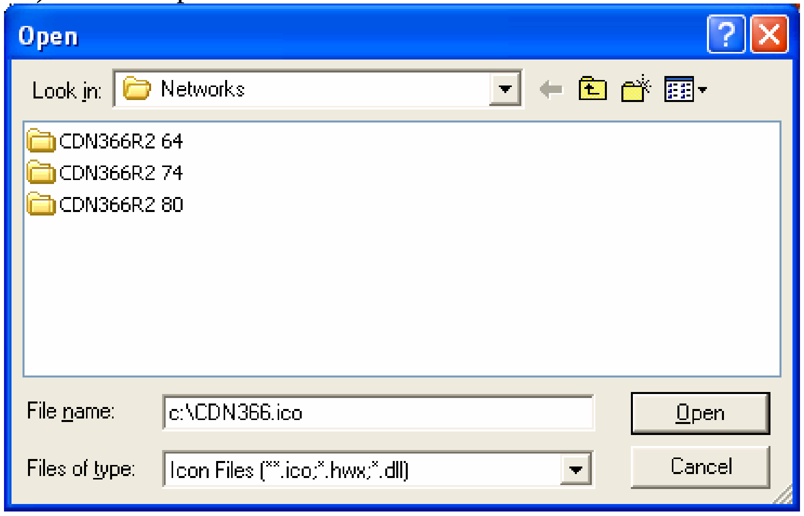
Change Graphic Image
You can change the graphic image that is associated with a device.

Product Types
- Communication Adapter
  - CDN366

9) The Change Icon screen pops up. Click *Browse* to enter path for CDN366 icon file. You can download the icon file from [www.mksinst.com.](http://www.mksinst.com)



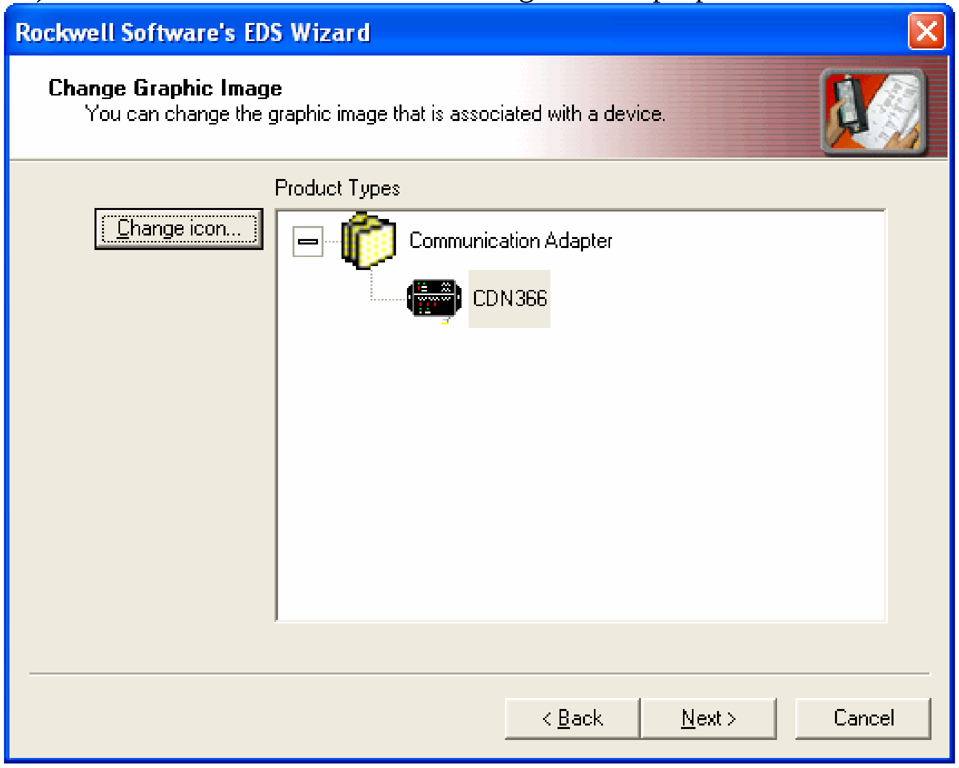
10) Enter the path to CDN366 icon file in the *File name:* box. Click *Open* to continue.



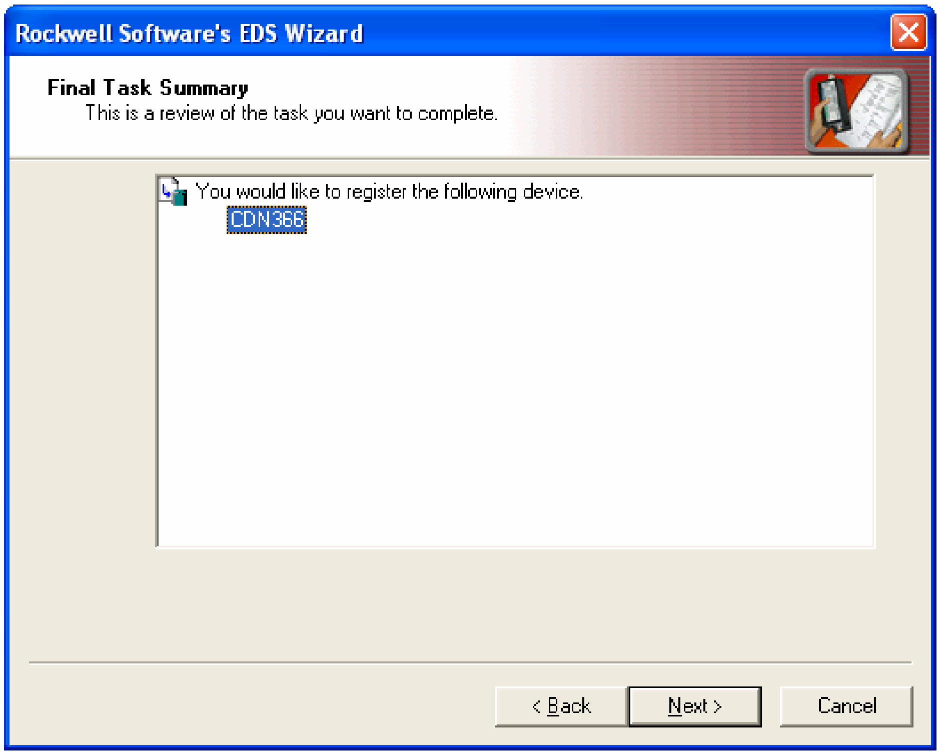
File name: c:\CDN366.ico

Files of type: Icon Files (*.ico;*.hwx;*.dll)

11) The CDN366 icon should have changed to the proper icon. Click *Next* to continue.

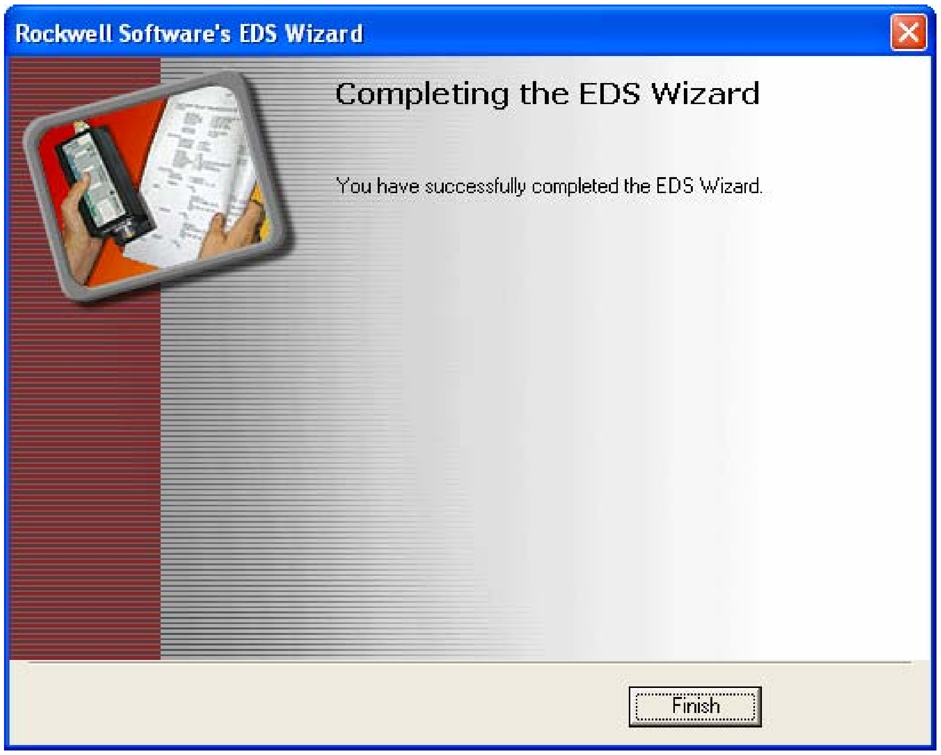

Rockwell Software's EDS Wizard
Change Graphic Image
You can change the graphic image that is associated with a device.
Product Types
Change icon...
Communication Adapter
CDN366
< Back
Next >
Cancel

12) The final step is to finish EDS file registration. Click *Next>* to complete the registration process. Click *Finish* to close the EDS Wizard window.



Final Task Summary
This is a review of the task you want to complete.

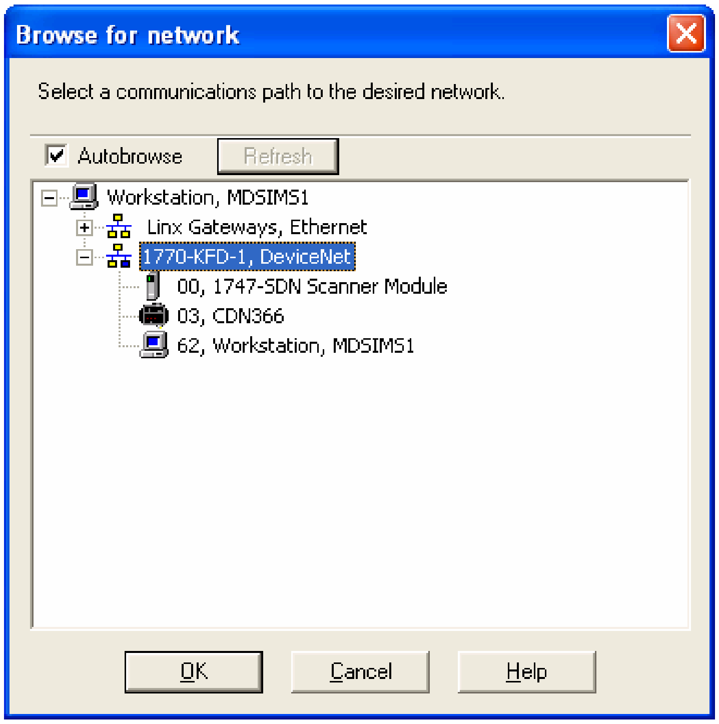
You would like to register the following device.
CDN366



Completing the EDS Wizard

You have successfully completed the EDS Wizard.

13) Repeat steps 1, 2, and 3 to browse the DeviceNet network. RSNetworx should now recognize the device at MAC ID 03 as a CDN366 gateway, and display the CDN366 icon. Click *Cancel* when finished.


Browse for network

Select a communications path to the desired network.

☑ Autobrowse

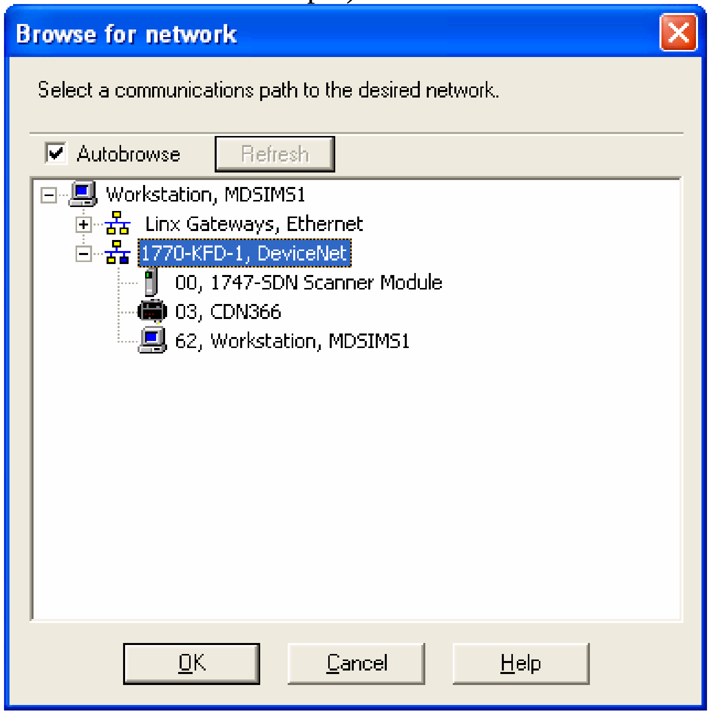
Workstation, MDSIMS1
Linx Gateways, Ethernet
1770-KFD-1, DeviceNet
00, 1747-SDN Scanner Module
03, CDN366
62, Workstation, MDSIMS1

Configure Serial Channel

Once the gateway is connected to DeviceNet and communicating with RSNetworx, you can configure its serial channel. Make sure the gateway is not in the DeviceNet master scanlist before changing any attribute values.

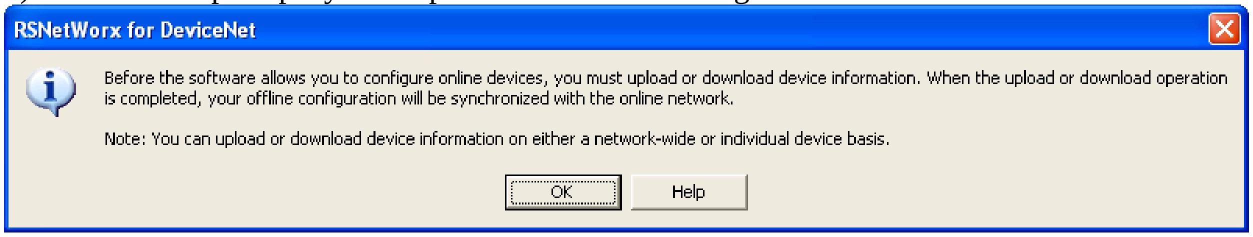
The Serial Stream Object attributes control the physical layer settings for the gateway's serial channel. The following steps show how to configure the Serial Stream Object attributes using the RSNetworx program.

1) Select the *Online* operation from the *Network* menu. Select the DeviceNet adapter (*1770-KFD-1* in this example) and click *OK*.


Browse for network

Select a communications path to the desired network.

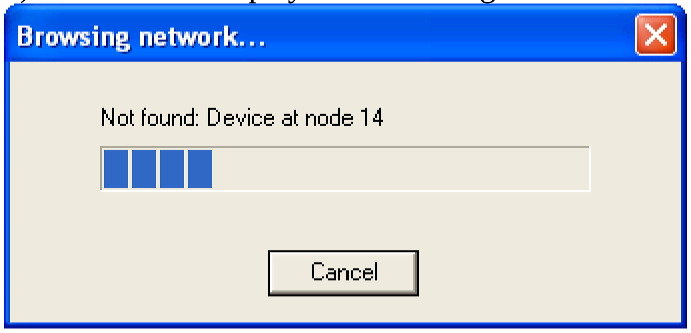
2) RSNetworx prompts you to upload the network configuration. Click *OK* to continue.


RSNetWorx for DeviceNet

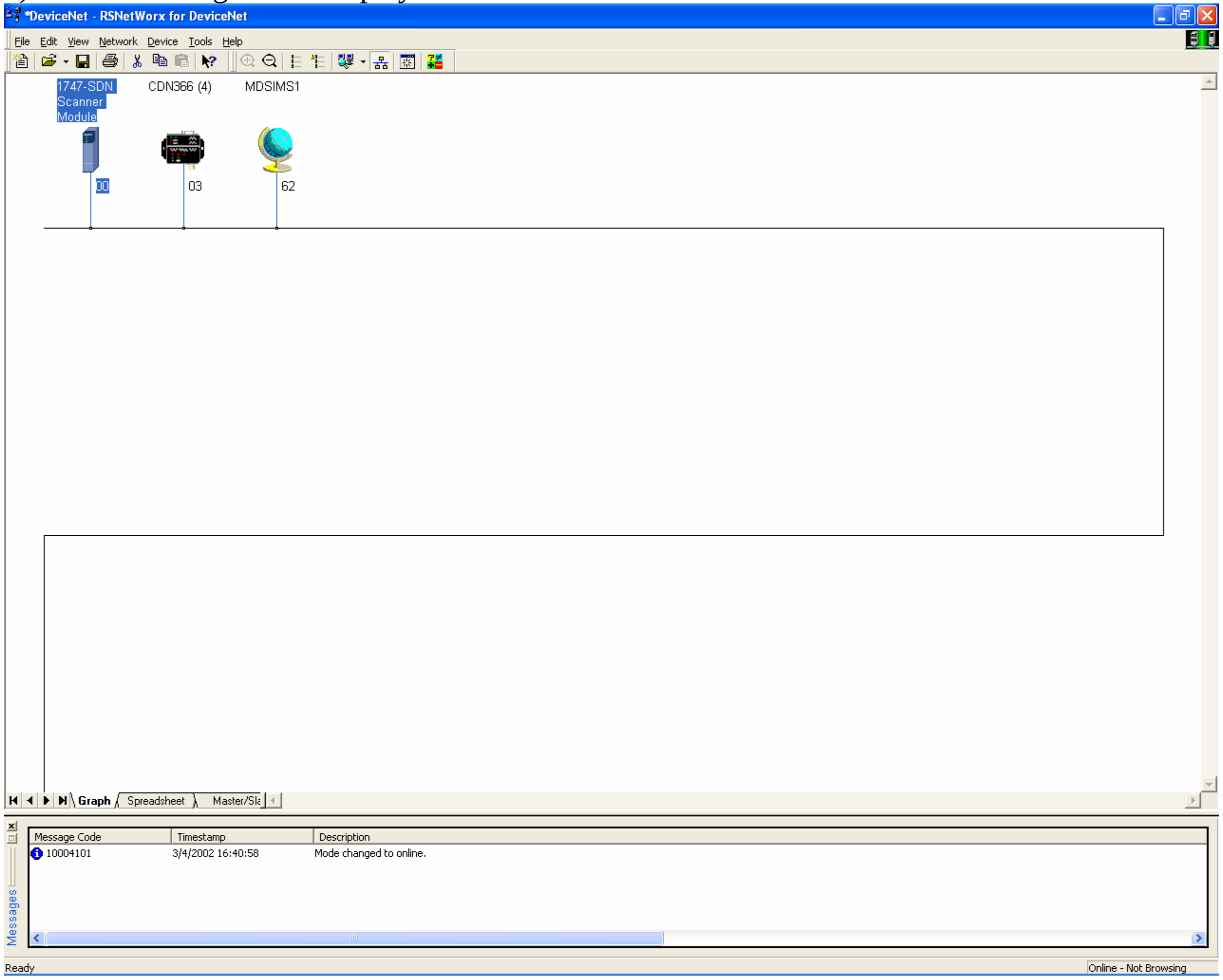
Before the software allows you to configure online devices, you must upload or download device information. When the upload or download operation is completed, your offline configuration will be synchronized with the online network.

Note: You can upload or download device information on either a network-wide or individual device basis.

3) RSNetworx displays the following text box while it uploads the network configuration.

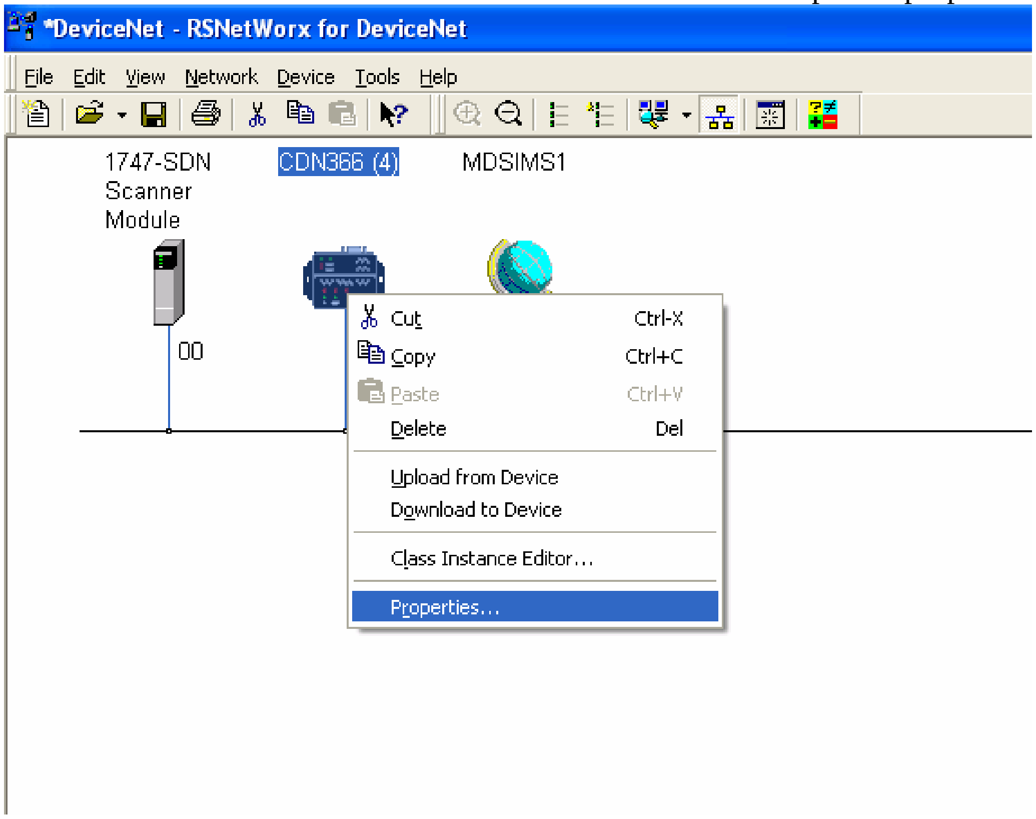


4) The following screen displays the online nodes.

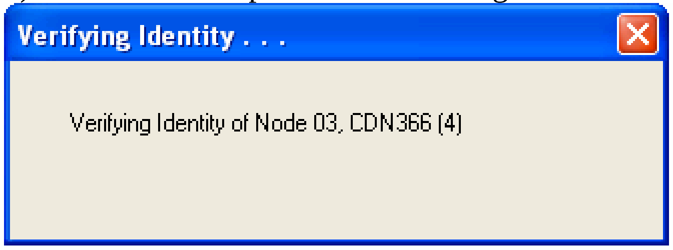


Message Code	Timestamp	Description
10004101	3/4/2002 16:40:58	Mode changed to online.

5) Left-click on the *CDN366 icon* to select it. Right-click and select *Properties* from the pop-up menu. You can also double-click on the *CDN366 icon* to open its properties box.

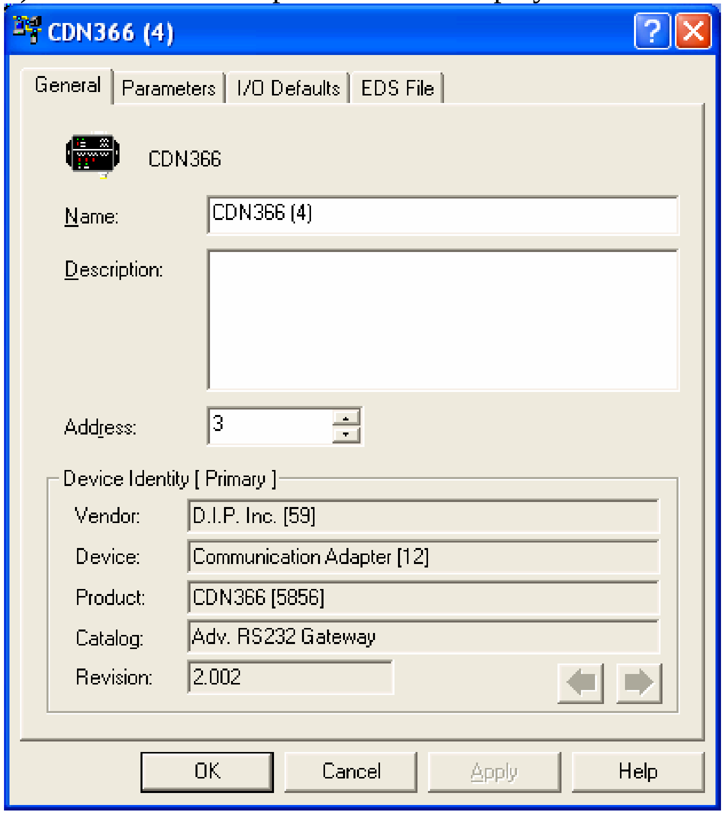


6) RSNetworx displaces the following text box while is reads CDN366 EDS file.



Verifying Identity of Node 03, CDN366 (4)

7) The CDN366 Properties Box is displayed.



CDN366

Name: CDN366 (4)

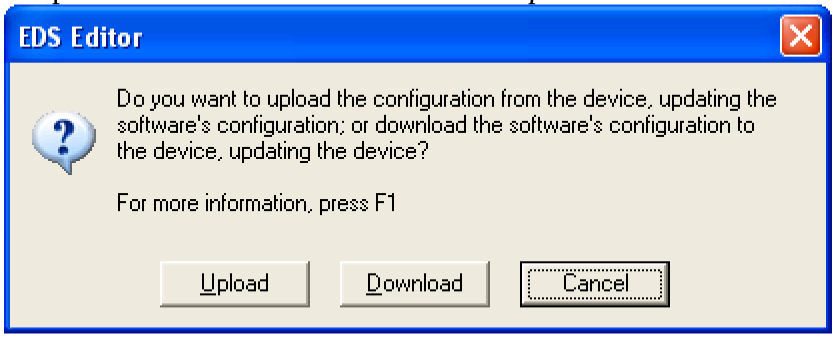
Description:

Address: 3

Device Identity [Primary]

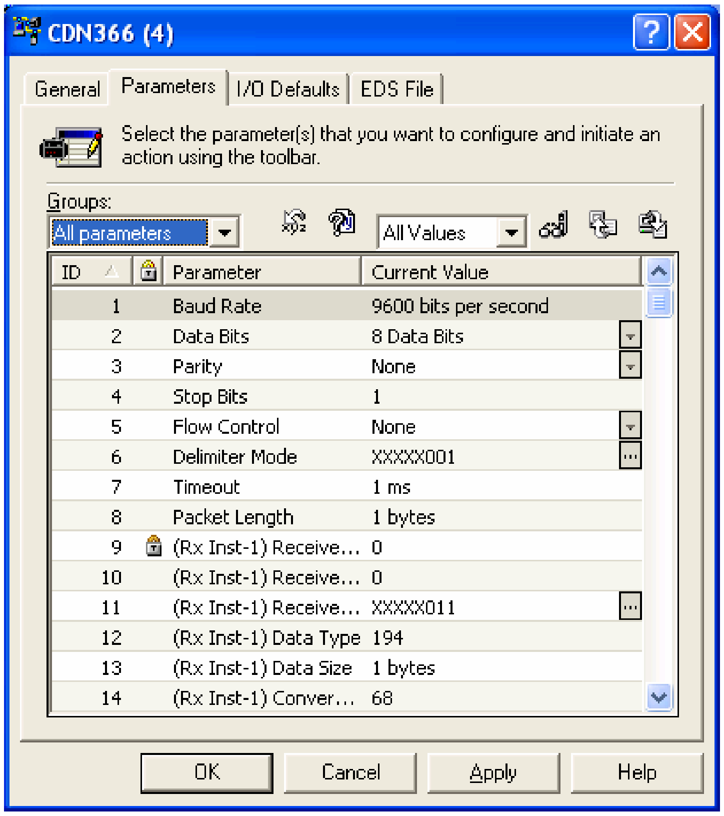
Vendor:	D.I.P. Inc. [59]
Device:	Communication Adapter [12]
Product:	CDN366 [5856]
Catalog:	Adv. RS232 Gateway
Revision:	2.002

7) Select the *Parameters* tab. You will be prompted for the parameters source. Select the *Upload* button to upload CDN366 parameters from the actual device. All the CDN366 parameters are now shown in the *Properties* window.



Do you want to upload the configuration from the device, updating the software's configuration; or download the software's configuration to the device, updating the device?

For more information, press F1



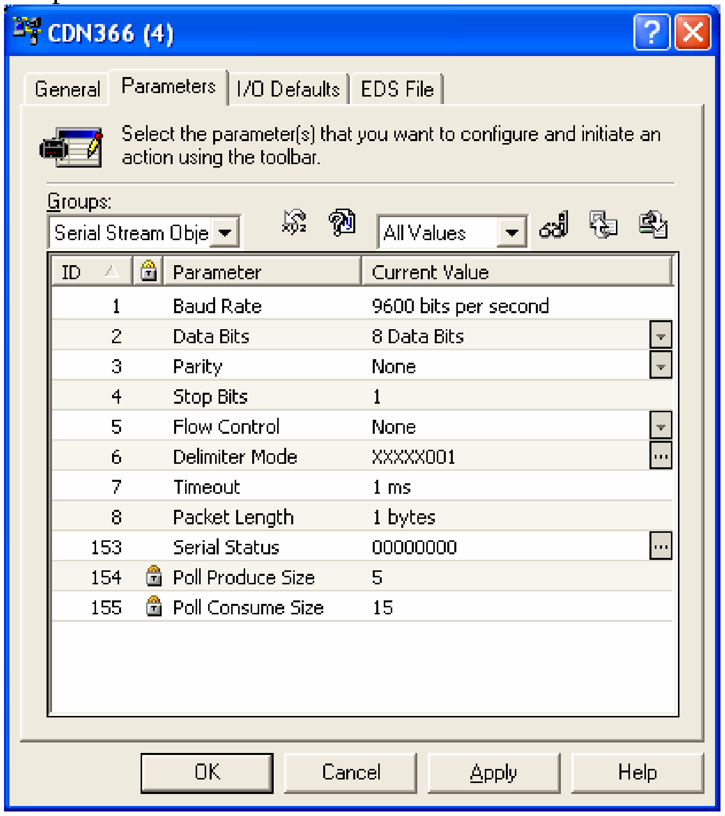
Select the parameter(s) that you want to configure and initiate an action using the toolbar.

Groups: All parameters

ID		Parameter	Current Value
1		Baud Rate	9600 bits per second
2		Data Bits	8 Data Bits
3		Parity	None
4		Stop Bits	1
5		Flow Control	None
6		Delimiter Mode	XXXXX001
7		Timeout	1 ms
8		Packet Length	1 bytes
9	🔒	(Rx Inst-1) Receive…	0
10		(Rx Inst-1) Receive…	0
11		(Rx Inst-1) Receive…	XXXXX011
12		(Rx Inst-1) Data Type	194
13		(Rx Inst-1) Data Size	1 bytes
14		(Rx Inst-1) Conver…	68

The CDN36X gateway has three Object types. The Serial Stream Object is used to configure the serial channel physical interface. This object will be configured in this section. The Serial Receive Object and Serial Transmit Object are in the next sections.

8) Select the *Serial Stream Object* from the *Groups* pull-down menu to view this object's parameters.



Select the parameter(s) that you want to configure and initiate an action using the toolbar.

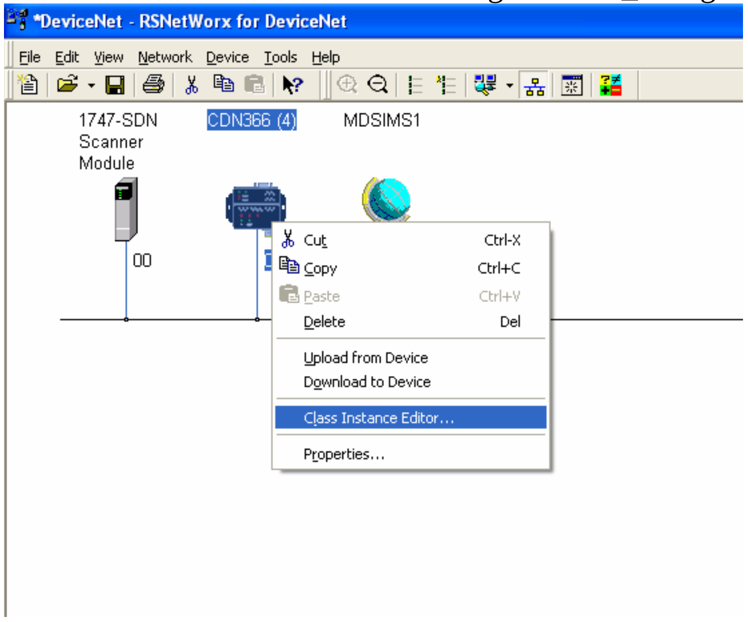
Groups: Serial Stream Obje

All Values

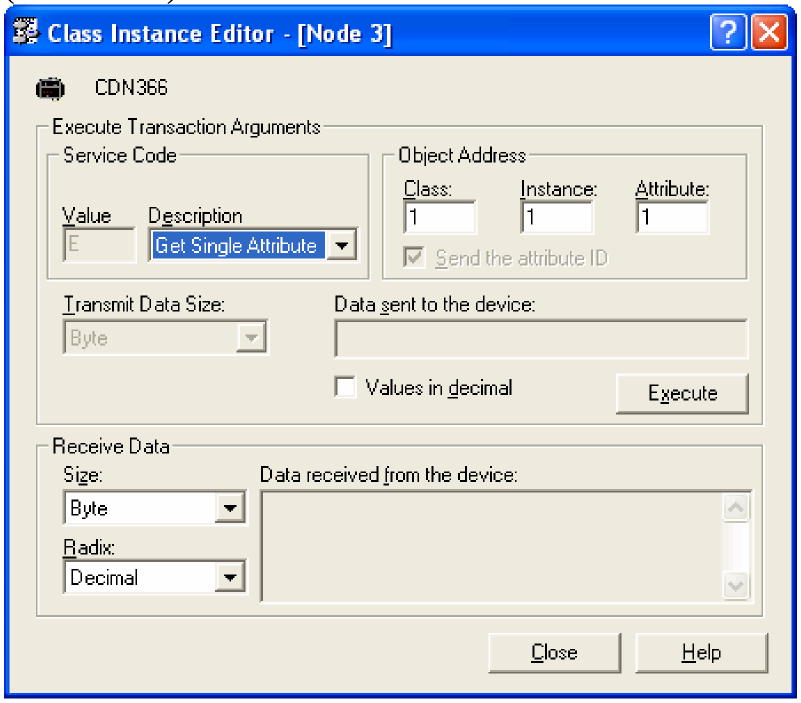
ID		Parameter	Current Value
1		Baud Rate	9600 bits per second
2		Data Bits	8 Data Bits
3		Parity	None
4		Stop Bits	1
5		Flow Control	None
6		Delimiter Mode	XXXXX001
7		Timeout	1 ms
8		Packet Length	1 bytes
153		Serial Status	00000000
154	🔒	Poll Produce Size	5
155	🔒	Poll Consume Size	15

You may now edit the Serial Stream Object attributes in this window.

Note that the **Pre-Delimiter List** and **Post-Delimiter List** attributes are not listed. These attributes use Short_String data type, which is not supported by RSNetworx EDS File interface. Use the Class Instance Editor to configure Short_String attributes.



Select the Set_Attribute_Single service code to write an attribute value, and the Get_Attribute_Single service code to read an attribute value. Check *Values in decimal* box to enter class, instance, attribute, and data values in decimal. The Pre-Delimiter List address is Class 64, Instance 1 (in this example), Attribute Number 11. The Post-Delimiter List address is Class 64, Instance 1 (in this example), Attribute Number 12. Enter the Short_String data as length byte, then data bytes. Example is [0x01 0x02] for 1-byte Pre-Delimiter List of 0x02 (ASCII STX).


Class Instance Editor - [Node 3]

CDN366

Execute Transaction Arguments

Service Code

Value: E

Description: Get Single Attribute

Object Address

Class: 1

Instance: 1

Attribute: 1

☑ Send the attribute ID

Transmit Data Size: Byte

Data sent to the device:

☐ Values in decimal

Receive Data

Size: Byte

Radix: Decimal

Data received from the device:

Enter the remaining Serial Stream Object attributes in the Parameters Box window.

Baud Rate – Click on the *current value* to change the baud rate. Enter the desired value in bits-per-second as a decimal number.

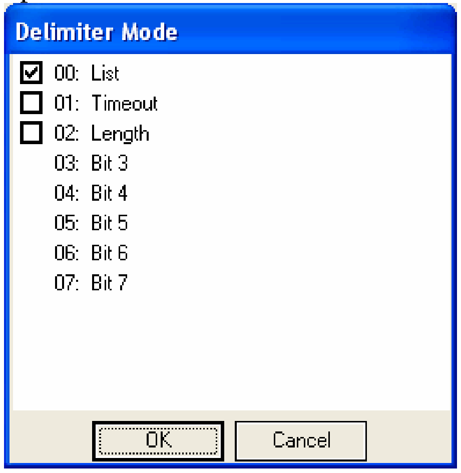
Data Bits – Click on arrow to the right of the *current value* to select from pull-down menu.

Stop Bits – Click on the *current value* to change. Enter the desired number of stop bits as a decimal number.

Parity – Click on arrow to the right of the *current value* to select from pull-down menu.

Flow Control – Click on arrow to the right of the *current value* to select from pull-down menu.

Delimiter Mode – Click on [...] to open selection box. Click on check box to select the desired option. The Delimiter Mode uses bits 0, 1, 2 of the byte. Ignore the remaining bits 3 through 7.



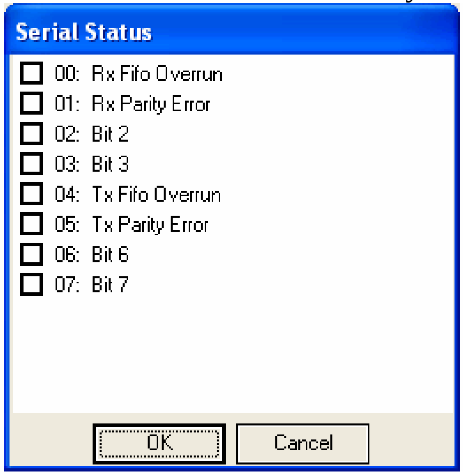
Delimiter Mode

- [x] 00: List
- [ ] 01: Timeout
- [ ] 02: Length
- 03: Bit 3
- 04: Bit 4
- 05: Bit 5
- 06: Bit 6
- 07: Bit 7

Timeout – Click on *current value* to change. Enter the desired timeout in milliseconds.

Packet Length – Click on *current value* to change. Enter the desired length in bytes.

Serial Status_– Click on [...] to open up selection box. Click on check box to set or clear the desired bit. The Serial Status byte uses bits 0, 1, 4, 5. Ignore the remaining bits 2, 3, 6, 7.



Serial Status

- [ ] 00: Rx Fifo Overrun
- [ ] 01: Rx Parity Error
- [ ] 02: Bit 2
- [ ] 03: Bit 3
- [ ] 04: Tx Fifo Overrun
- [ ] 05: Tx Parity Error
- [ ] 06: Bit 6
- [ ] 07: Bit 7

The serial status bits are set (bit = 1) by the CDN36X gateway when an error occurs. You must acknowledge the receipt of an error by clearing the appropriate bit (bit = 0). Clearing an error bit causes the gateway to clear that error condition and resume normal operation. You must either reset the CDN36X or clear each error bit using a Set_Attribute explicit message command in order to resume normal operation.

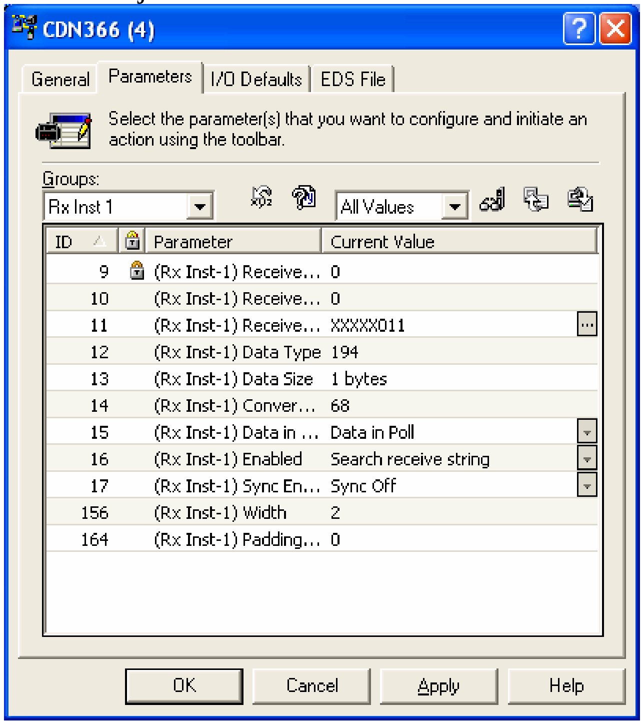
I/O Produce Size – This is a read only attribute. The value is the number of data bytes the gateway returns in the I/O Response Message.

I/O Consume Size – This is a read only attribute. The value is the number of data bytes the gateway expects to receive in the I/O Command Message.

Configure Serial Receive Object Instances

There are eight identical Serial Receive Object instance parameter sets that can be configured in the CDN366 gateway. The following describes how to configure Serial Receive Object Instance 1. Program the other instances using the same procedure.

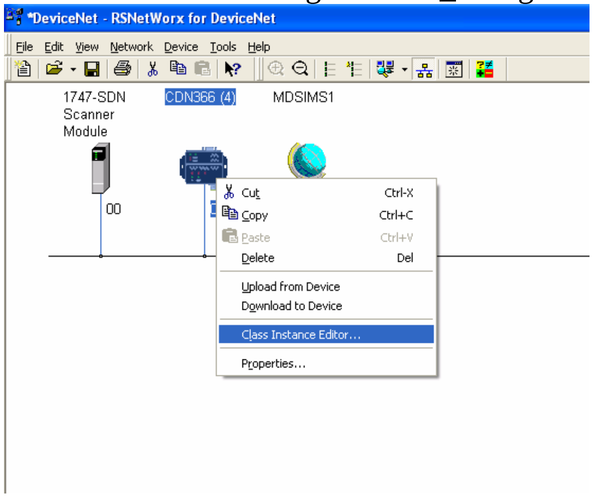
1) Using RSNetworx, open the CDN366 gateway *Properties* window. Select the *Parameters* tab. Select *Rx Inst 1* from the *Groups* pull-down menu. You show see the 10 attributes for this object instance.


CDN366 (4)
General
Parameters
I/O Defaults
EDS File
Select the parameter(s) that you want to configure and initiate an action using the toolbar.
Groups:
Rx Inst 1
All Values
OK
Cancel
Apply
Help

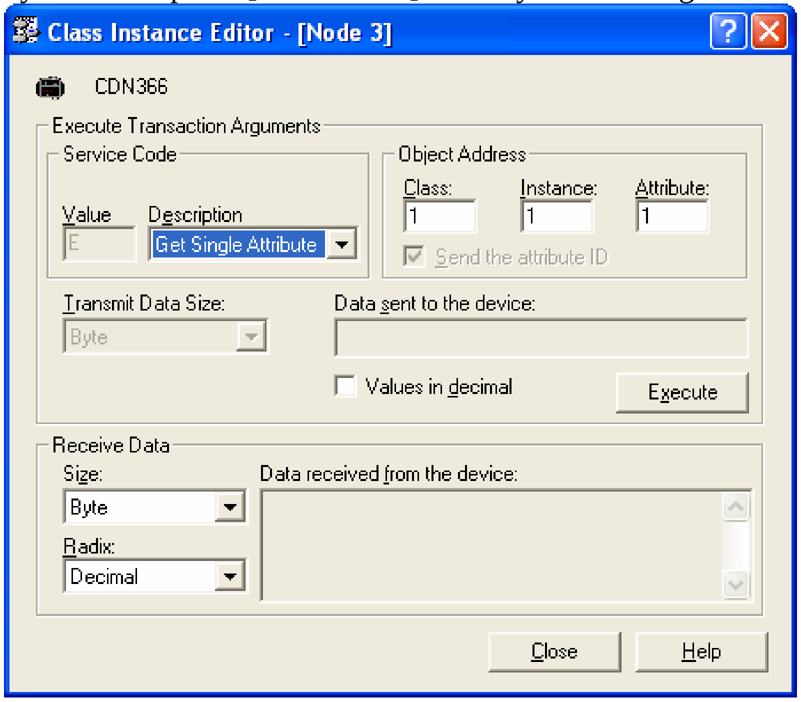
ID	🔒	Parameter	Current Value
9	🔒	(Rx Inst-1) Receive…	0
10		(Rx Inst-1) Receive…	0
11		(Rx Inst-1) Receive…	XXXXX011
12		(Rx Inst-1) Data Type	194
13		(Rx Inst-1) Data Size	1 bytes
14		(Rx Inst-1) Conver…	68
15		(Rx Inst-1) Data in …	Data in Poll
16		(Rx Inst-1) Enabled	Search receive string
17		(Rx Inst-1) Sync En…	Sync Off
156		(Rx Inst-1) Width	2
164		(Rx Inst-1) Padding…	0

You may now edit the Serial Receive Object Instance 1 attributes in this window.

Note that the **Pre-String** and **Post-String** attributes are not listed. These attributes use Short_String data type, which is not supported by RSNetworx EDS File interface. Use the Class Instance Editor to configure Short_String attributes.



Select the Set_Attribute_Single service code to write an attribute value, and the Get_Attribute_Single service code to read an attribute value. Check *Values in decimal* box to enter class, instance, attribute, and data values in decimal. The Pre-String address is Class 65, Instance 1 (in this example), Attribute Number 7. The Post-String address is Class 65, Instance 1 (in this example), Attribute Number 8. Enter the Short_String data as length byte, then data bytes. Example is [0x01 0x41] for 1-byte Pre-String of 0x41 (ASCII 'A').


Class Instance Editor - [Node 3]

CDN366

Execute Transaction Arguments

Service Code

Value: E

Description: Get Single Attribute

Object Address

Class: 1

Instance: 1

Attribute: 1

[✓] Send the attribute ID

Transmit Data Size: Byte

Data sent to the device:

[ ] Values in decimal

Execute

Receive Data

Size: Byte

Radix: Decimal

Data received from the device:

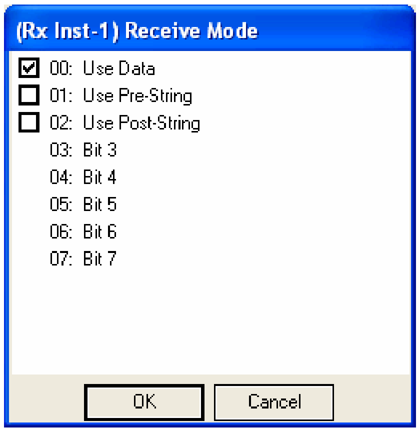
Close Help

Enter the remaining Serial Receive Object attributes in the Parameters Box.

Receive Toggle – Read only attribute. Bit toggled (0 to 1, or 1 to 0) when a new data packet has been received, indicating that it is ready to be read as DeviceNet inputs.

Receive Acknowledge – Click on current value to change. Enter 0 to clear, 1 to set.

Receive Mode – Click on ... to open up selection box. Click on check box to set or clear the desired bit. The Receive Mode bits are 0, 1, 2. Ignore the remaining bits 3 through 7.


(Rx Inst-1) Receive Mode

[✓] 00: Use Data

[ ] 01: Use Pre-String

[ ] 02: Use Post-String

03: Bit 3

04: Bit 4

05: Bit 5

06: Bit 6

07: Bit 7

OK Cancel

Data Type – Click on *current value* to change. Enter decimal number of desired data type.

Data Size – Click on *current value* to change. Enter the desired Short_String data size in bytes (2-128). Do not enter a Data Size for integer or real number Data Types.

Width – Click on *current value* to change. Enter expected width as decimal number (1-16).

Conversion Type – Click on *current value* to change. Enter ‘D’ for decimal and ‘X’ for hex.

Data in I/O Response – Click on *current value* and select from pull-down menu.

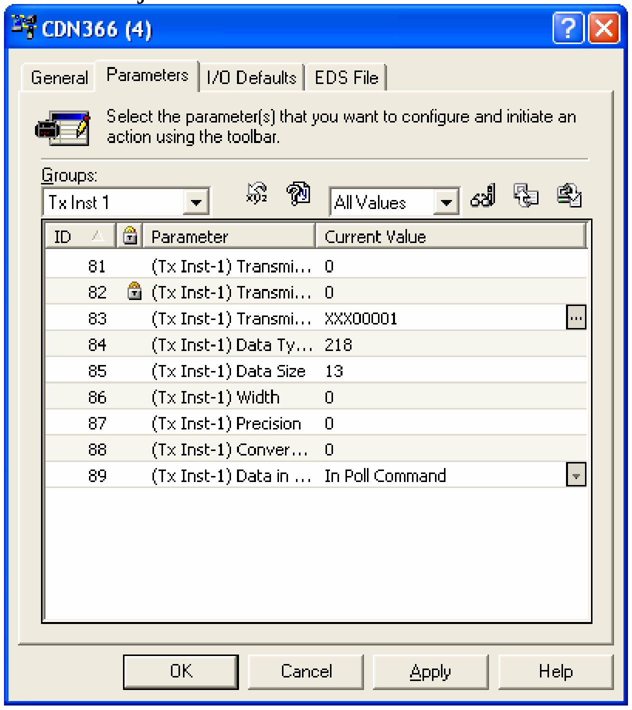
Enabled – Click on *current value* and select from pull-down menu. Select *Ignore this instance* to disable the Serial Receive Object instance. Select *Search receive string* to enable the Serial Receive Object instance.

Sync Enabled – Click on *current value* and select from pull-down menu. Select *Sync Off* to disable synchronization. Select *Sync On* to enable synchronization.

Configure Serial Transmit Object Instances

There are eight identical Serial Transmit Object instance parameter sets that can be configured in the CDN366 gateway. The following describes how to configure Serial Transmit Object Instance 1. Program the other instances following the same procedure.

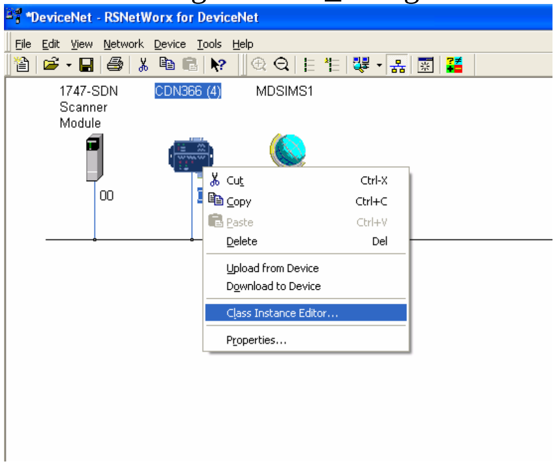
1) Using RSNetWorx, open the CDN366 gateway *Properties* window. Select the *Parameters* tab. Select *Tx Inst 1* from the *Groups* pull-down menu. You show see the 9 attributes for this object instance.


CDN366 (4)
General
Parameters
I/O Defaults
EDS File
Select the parameter(s) that you want to configure and initiate an action using the toolbar.
Groups:
Tx Inst 1
All Values
OK
Cancel
Apply
Help

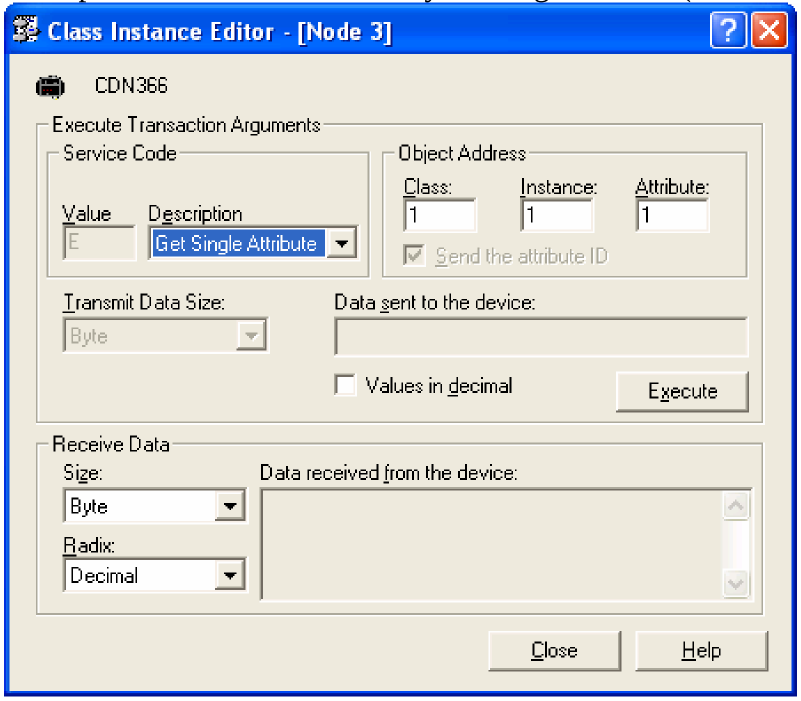
ID		Parameter	Current Value
81		(Tx Inst-1) Transmi…	0
82	🔒	(Tx Inst-1) Transmi…	0
83		(Tx Inst-1) Transmi…	XXX00001
84		(Tx Inst-1) Data Ty…	218
85		(Tx Inst-1) Data Size	13
86		(Tx Inst-1) Width	0
87		(Tx Inst-1) Precision	0
88		(Tx Inst-1) Conver…	0
89		(Tx Inst-1) Data in …	In Poll Command

You may now edit the Serial Receive Object Instance 1 attributes in this window.

Note that the **String1** and **String2** attributes are not listed. These attributes use Short_String data type, which is not supported by RSNetworx EDS File interface. Use the Class Instance Editor to configure Short_String attributes.



Select the Set_Attribute_Single service code to write an attribute value, and the Get_Attribute_Single service code to read an attribute value. Check *Values in decimal* box to enter class, instance, attribute, and data values in decimal. The String1 address is Class 66, Instance 1 (in this example), Attribute Number 7. The String2 address is Class 66, Instance 1 (in this example), Attribute Number 8. Enter the Short_String data as length byte, then data bytes. Example is [0x01 0x41] for 1-byte String1 of 0x41 (ASCII 'A').


Class Instance Editor - [Node 3]

CDN366

Execute Transaction Arguments

Service Code

Value: E

Description: Get Single Attribute

Object Address

Class: 1

Instance: 1

Attribute: 1

[✓] Send the attribute ID

Transmit Data Size: Byte

Data sent to the device:

[ ] Values in decimal

Execute

Receive Data

Size: Byte

Radix: Decimal

Data received from the device:

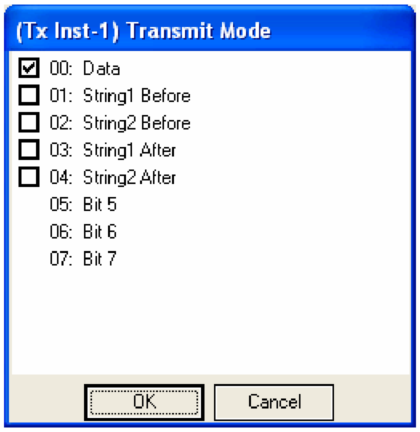
Close Help

Enter the remaining Serial Transmit Object attributes in the Parameters Box.

Transmit Toggle – Click on current value to change. Enter 0 to clear, 1 to set.

Transmit Acknowledge – Read only attribute. Bit toggled (0 to 1, or 1 to 0) after Instance loads serial message packet into the transmit buffer, and is ready for the next message.

Transmit Mode – Click on ... to open up selection box. Click on check box to set or clear the desired bit. The Transmit Mode bits are 0, 1, 2, 3, 4. Ignore the remaining bits 5, 6, 7.


(Tx Inst-1) Transmit Mode

[✓] 00: Data

[ ] 01: String1 Before

[ ] 02: String2 Before

[ ] 03: String1 After

[ ] 04: String2 After

05: Bit 5

06: Bit 6

07: Bit 7

OK Cancel

Data Type – Click on *current value* to change. Enter decimal number of desired data type.

Data Size – Click on *current value* to change. Enter the desired Short_String data size in bytes (2-128). Do not enter a Data Size for integer or real number Data Types.

Width – Click on *current value* to change. Enter expected width as decimal number (1-16).

Conversion Type – Click on *current value* to change. Enter decimal number (see Ch 4).

Precision – Click on *current value* to change. Enter desired precision in decimal (0-6).

Data in I/O Command – Click on *current value* and select from pull-down menu.

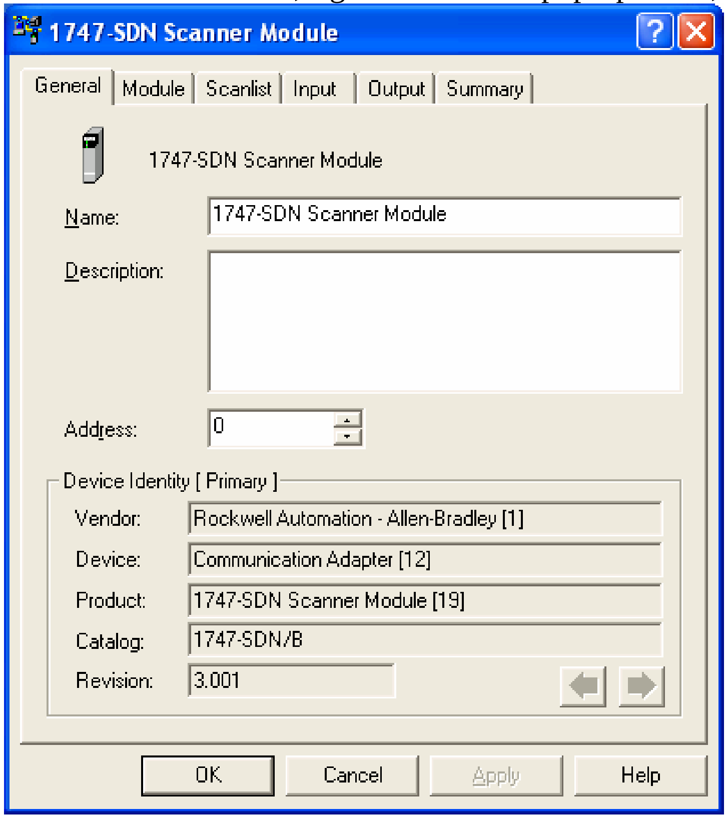
Configure DeviceNet Master Scanlist

After all the object instances have been configured, the DeviceNet master can be configured for Polled I/O or Change-of-State I/O messaging with the gateway. The following example shows how to configure a Polled I/O connection.

1) Before using the RSNetworx to map the gateway's I/O connection to 1747-SDN DeviceNet master scanner, you must calculate the *I/O Produce Size* & *I/O Consume Size*. Chapter 4 describes how to calculate these values. You can also read them directly from the gateway Serial Stream Object *I/O Product Size* and *I/O Consume Size* attributes. Follow the steps in ***Configure Serial Channel*** section to read these attribute values.

2) Match sure all unused Serial Receive Object and Serial Transmit Object instances are disabled. Follow the steps in ***Configure Serial Receive Object Instances*** and ***Configure Serial Transmit Object Instances*** sections to disable unused instances.

3) Double click on the 1747-SDN icon to open its *Properties* box. You can also left click on the icon to select it, right click for the pop-up menu, and select *Properties*.



1747-SDN Scanner Module

Name: 1747-SDN Scanner Module

Description:

Address: 0

Device Identity [Primary]

Vendor: Rockwell Automation - Allen-Bradley [1]

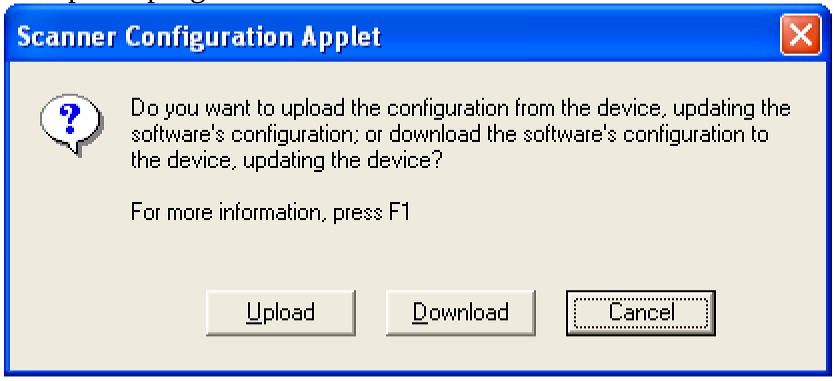
Device: Communication Adapter [12]

Product: 1747-SDN Scanner Module [19]

Catalog: 1747-SDN/B

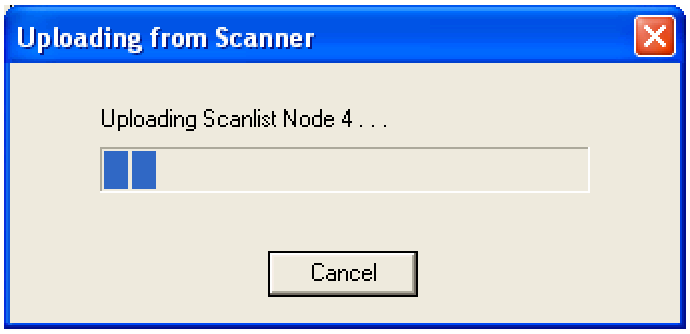
Revision: 3.001

4) Select the Scanlist tab. RSNetworx prompts you for the Scanner Configuration. Click Upload to upload current 1747-SDN configuration from the node. RSNetworx displays the upload progress.


Scanner Configuration Applet

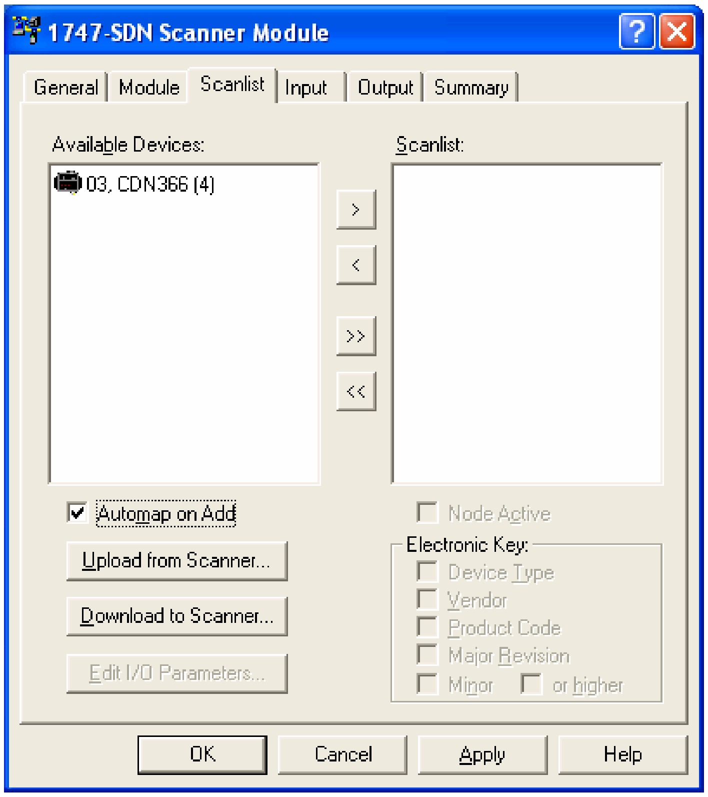
Do you want to upload the configuration from the device, updating the software's configuration; or download the software's configuration to the device, updating the device?

For more information, press F1


Uploading from Scanner

Uploading Scanlist Node 4 . . .

5) The next window shows the *Available Devices:* that can be added to the 1747-SDN *Scanlist*.



Available Devices:
03, CDN366 (4)

Scanlist:

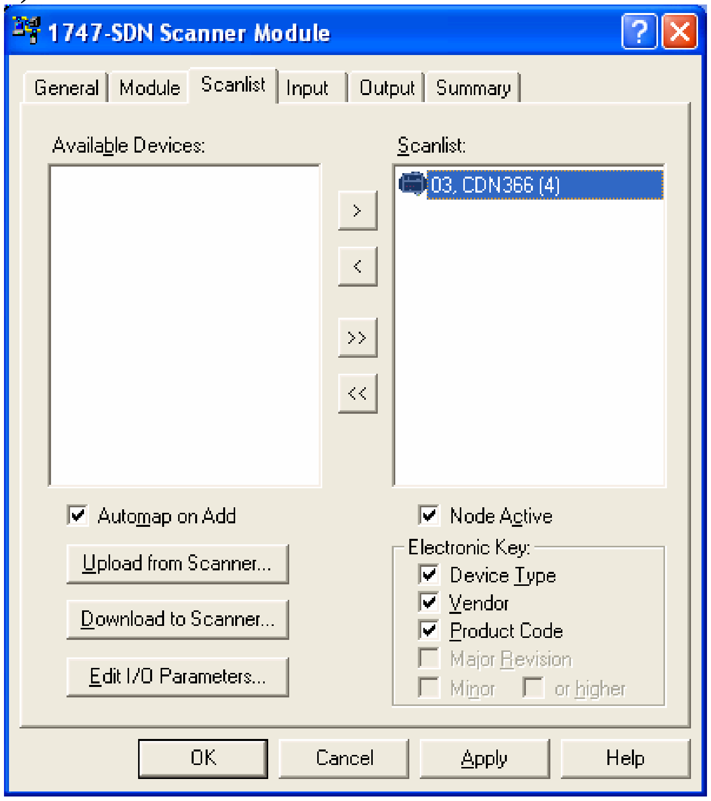
☑ Automap on Add

☐ Node Active

Electronic Key:
☐ Device Type
☐ Vendor
☐ Product Code
☐ Major Revision
☐ Minor ☐ or higher

6) Select the Automap on Add checkbox if you want RSNetworx to automatically map the CDN366 input and output bytes into the 1747-SDN memory.

7) Select the CDN366 under *Available Devices:* and click the $\boxed{>}$ button to transfer to *Scanlist*.



Available Devices:

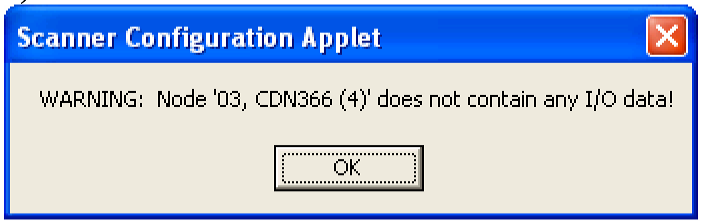
Scanlist:
03, CDN366 (4)

☑ Automap on Add

☑ Node Active

Electronic Key:
☑ Device Type
☑ Vendor
☑ Product Code
☐ Major Revision
☐ Minor ☐ or higher

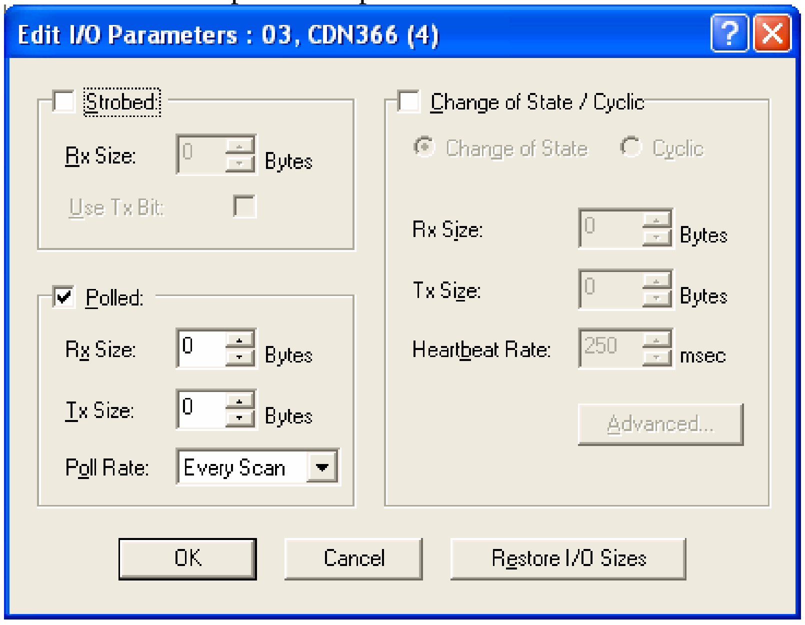
8) RSNetworx warns that the CDN366 does not contain any I/O data. Click *OK* to continue.



Scanner Configuration Applet

WARNING: Node '03, CDN366 (4)' does not contain any I/O data!

8) Click on the *Edit I/O Parameters* button. Use the $\boxed{\wedge}$ and $\boxed{\vee}$ buttons to set *Rx Size:* to the calculated I/O Consume Size value and the *Tx Size:* to the calculated I/O Produce Size value. Click *OK* to update I/O parameters.



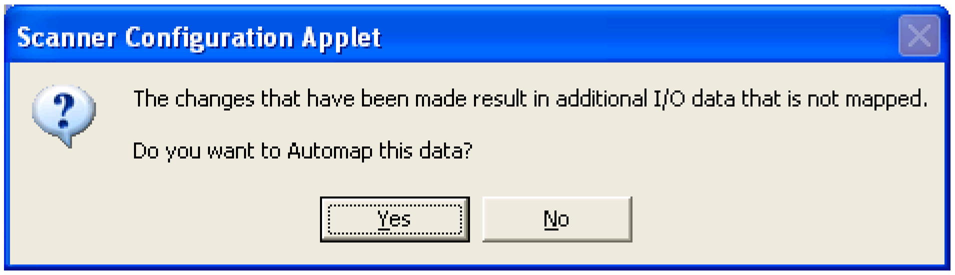
Edit I/O Parameters : 03, CDN366 (4)

[ ] Strobed:
Rx Size: 0 Bytes
Use Tx Bit: [ ]

[✓] Polled:
Rx Size: 0 Bytes
Tx Size: 0 Bytes
Poll Rate: Every Scan

[ ] Change of State / Cyclic
(●) Change of State ( ) Cyclic
Rx Size: 0 Bytes
Tx Size: 0 Bytes
Heartbeat Rate: 250 msec

9) RSNetworx prompts to Automap the new input and output data bytes. Select *Yes* to automap. If you select *No*, then you must manually map the I/O bytes in the memory tables.

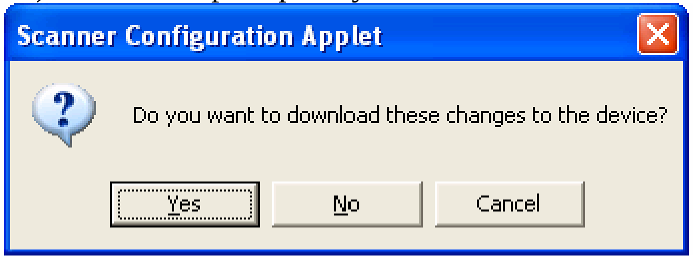


Scanner Configuration Applet

The changes that have been made result in additional I/O data that is not mapped.

Do you want to Automap this data?

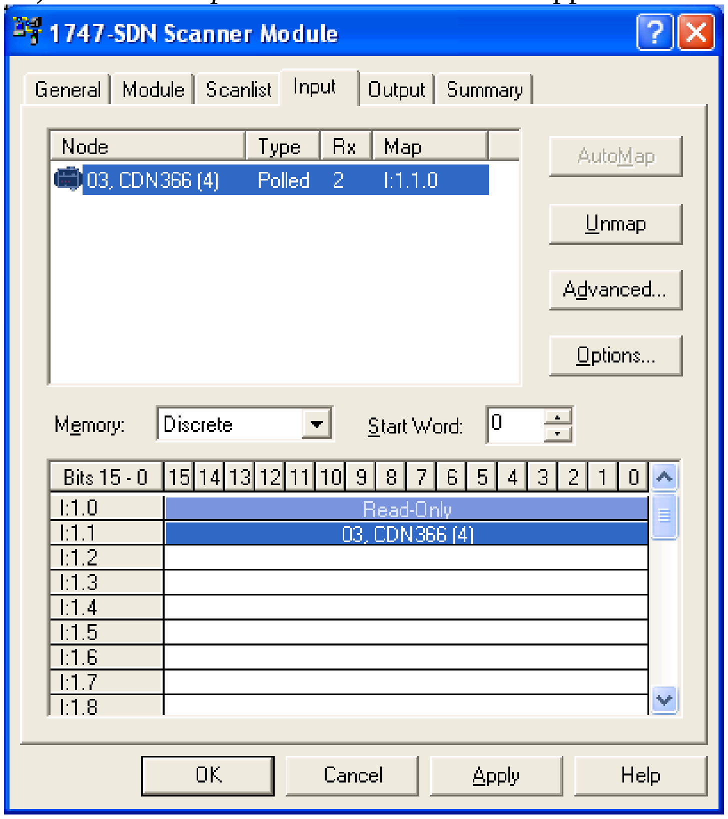
10) RSNetworx prompts if you want to download the changes to the 1747-SDN. Click *Yes*.



Scanner Configuration Applet

Do you want to download these changes to the device?

11) Select the *Input* tab to view the automapped CDN366 input bytes.



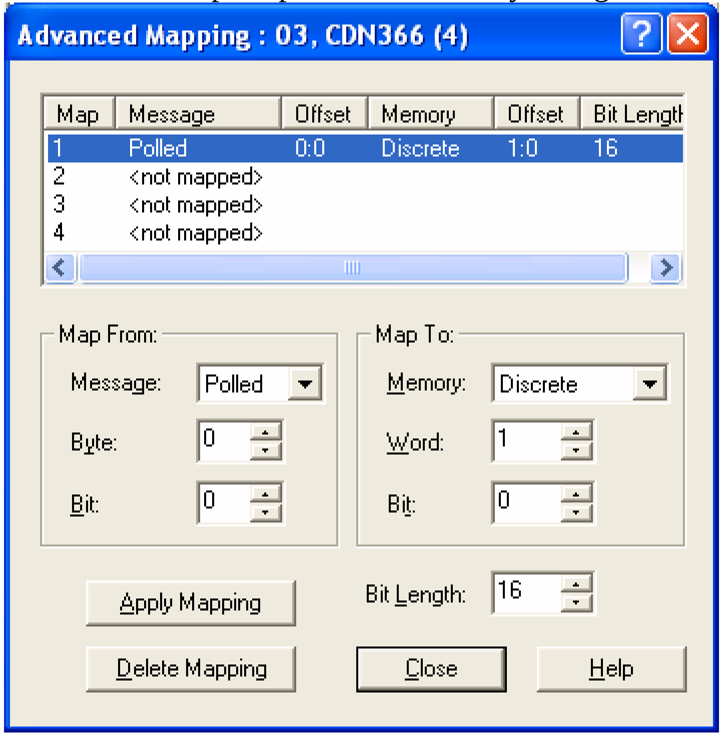
1747-SDN Scanner Module

Node	Type	Rx	Map
03, CDN366 (4)	Polled	2	I:1.1.0

Memory: Discrete   Start Word: 0

Bits 15 - 0	15-0
I:1.0	Read-Only
I:1.1	03, CDN366 (4)
I:1.2	
I:1.3	
I:1.4	
I:1.5	
I:1.6	
I:1.7	
I:1.8	

12) Click the *Advanced…* button to view current input mapping detail. Change the mapping to suit your application. Click *Apply Mapping* button after you make changes. Click *Yes* at the RSNetworx prompt to download any changes to the 1747-SDN. Click *Close* to continue.


Advanced Mapping : 03, CDN366 (4)

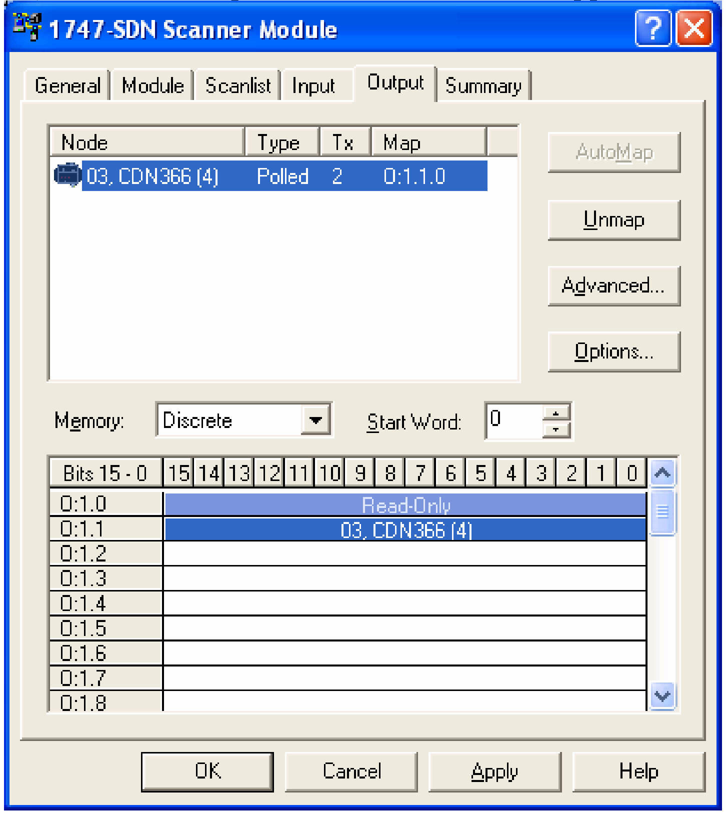
Map	Message	Offset	Memory	Offset	Bit Length
1	Polled	0:0	Discrete	1:0	16
2	<not mapped>				
3	<not mapped>				
4	<not mapped>				

Map From: Message: Polled; Byte: 0; Bit: 0

Map To: Memory: Discrete; Word: 1; Bit: 0

Bit Length: 16

13) Select the *Output* tab to view the automapped CDN366 output bytes.

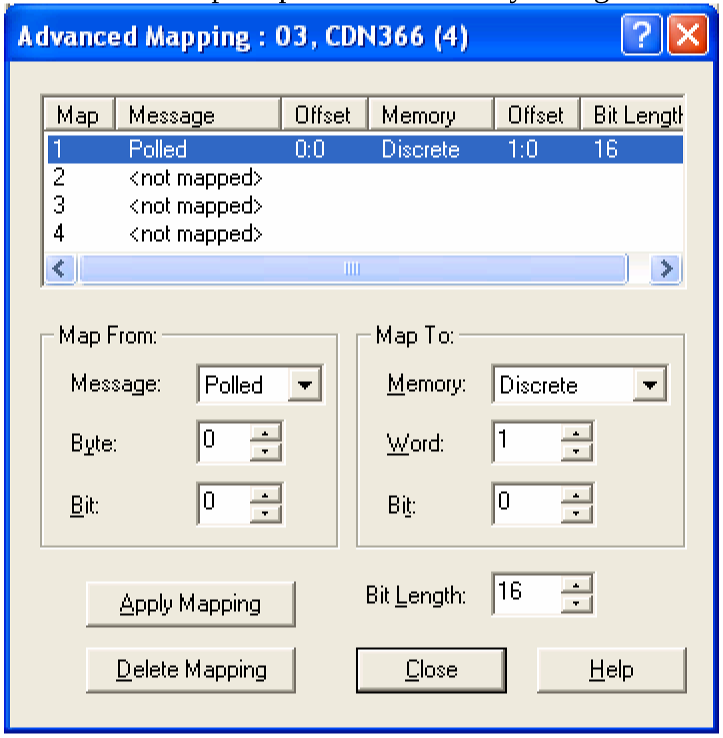

1747-SDN Scanner Module

Node	Type	Tx	Map
03, CDN366 (4)	Polled	2	0:1.1.0

Memory: Discrete Start Word: 0

Bits 15 - 0	15–0
0:1.0	Read-Only
0:1.1	03, CDN366 (4)
0:1.2	
0:1.3	
0:1.4	
0:1.5	
0:1.6	
0:1.7	
0:1.8	

14) Click the *Advanced…* button to view current input mapping detail. Change the mapping to suit your application. Click *Apply Mapping* button after you make changes. Click *Yes* at the RSNetworx prompt to download any changes to the 1747-SDN. Click *Close* to continue.



Advanced Mapping : 03, CDN366 (4)

Map	Message	Offset	Memory	Offset	Bit Length
1	Polled	0:0	Discrete	1:0	16
2	<not mapped>				
3	<not mapped>				
4	<not mapped>				

Map From:
Message: Polled
Byte: 0
Bit: 0

Map To:
Memory: Discrete
Word: 1
Bit: 0

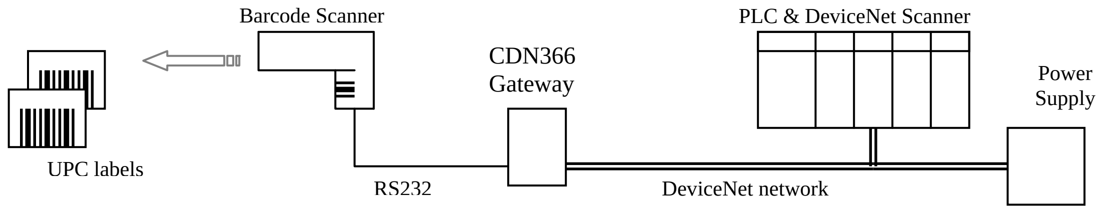
Bit Length: 16

Chapter 7 – Configuration Examples

This chapter contains four example gateway configurations.

Example 1 – Receiving Data

Read UPC labels into a PLC using a serial barcode scanner, a CDN366 gateway, and a DeviceNet scanner (master). The barcode scanner RS232 channel is connected to a CDN366 serial channel. The CDN366 DeviceNet channel is connected to the PLC DeviceNet scanner. The DeviceNet network is powered by an external 24VDC power supply.



Barcode Scanner

The barcode scanner's RS232 channel is set for 9600 bps, 8 data bits, no parity, and 1 stop bit. When it reads a UPC label, it transmits the following ASCII message format. The message always begins with the ASCII STX start-of-text (0x02) character, and always ends with the ASCII ETX end-of-text (0x03) and CR carriage return (0x0D) characters. The barcode data will consist of a variable number of 1 to 12 ASCII characters, depending upon the UPC label being scanned.

[STX] [ASCII barcode data] [ETX] [CR]

CDN366 Gateway

The CDN366 gateway needs to be configured to receive this RS232 message format. The first step is to determine the *Delimiter Mode*. The barcode scanner transmits a variable-length message packet, so *Length Mode* cannot be used. *Timeout Mode* may be used, but without specific timing information for the barcode scanner's RS232 channel it may be difficult to derive a suitable *Packet Timeout* value. *List Mode* is best suited for this application, because the serial message always begins and ends with the same characters. The *Serial Stream Object* can now be configured. The following shows the *Serial Stream Object* attribute settings for this application. The 3rd column lists the address string if using Set_Attribute_Single commands to write the attribute values. The last two attributes are Short_String data types.

***Serial Stream Object* Configuration (Class Code 64 or 0x40)**

Attribute	Data	Class / Instance / Attribute / Data	Description
3. Baud Rate	9600	0x40 0x01 0x03 0x25 0x80	9600 bps
4. Data Bits	8	0x40 0x01 0x04 0x08	8 data bits
5. Parity	0	0x40 0x01 0x05 0x00	no parity
6. Stop Bits	1	0x40 0x01 0x06 0x01	1 stop bit
7. Flow Control	2	0x40 0x01 0x07 0x02	CTS / RTS
10. Delimiter Mode	1	0x40 0x01 0x0A 0x01	List Mode
11. Pre-Delimiter String	0x01 0x02	0x40 0x01 0x0B 0x01 0x02	Short_String length = 1, STX
12. Post-Delimiter String	0x02 0x03 0x0D	0x40 0x01 0x0C 0x02 0x03 0x0D	Short_String length = 2, ETX CR

The next step is to configure the CDN366 gateway to return the ASCII barcode data to the DeviceNet scanner. Because the content of the ASCII bytes is not known, the entire byte string will be converted into a Short_String data type. With only one data variable to return, one *Serial Receive Object* Instance is configured. The *Data Type* is Short_String, with a *Data Size* of 13 (maximum number of expected barcode data bytes is 12, plus the length byte). The *Receive Mode* is Use Data Field.

***Serial Receive Object* Instance 1 Configuration (Class Code 65 or 0x41)**

Attribute	Data	Class / Instance / Attribute / Data	Description
6. Receive Mode	1	0x41 0x01 0x06 0x01	Use Data Field
9. Data Type	218	0x41 0x01 0x09 0xDA	Short_String
10. Data Size	13	0x41 0x01 0x0A 0x0D	1 length byte, 12 data bytes
15. Data in I/O Response	1	0x41 0x01 0x0F 0x01	Enable data in I/O response
16. Enabled	1	0x41 0x01 0x10 0x01	Instance 1 enabled

Make sure *Serial Receive Object* instances 2-8 are disabled, since only Instance 1 is used in this application. The gateway will return 15 input bytes to the DeviceNet scanner in the I/O Response Message. The *I/O Produce Size* is 15, with the data organized as follows:

```
[ Transmit Acknowledge bits ] [ Receive Toggle bits ] [ Instance 1 Short_String data ]
           1 byte                       1 byte                     13 bytes
```

The gateway will always return 13 bytes in the I/O Response Message, even if the scanned barcode data contains fewer bytes. The application should check the Short_String length byte to determine the number of valid data bytes being returned in a particular I/O Response Message.

Receive synchronization may also be used by enabling the *Sync Enabled* attribute. The *I/O Produce Size* can be verified by reading the *Serial Stream Object's I/O Produce Size* attribute (class 64, instance 1, attribute 20).

<u>Example 2 – Receiving Delimited Data</u>

Using the same configuration as Example 1, the scanned UPC labels are printed in one of two formats: [**MODEL xxx A**] and [**SN: xxxxx**]. The first format is a model number, and 'xxx' are 3 ASCII characters that represent a number from 1 to 100. The second format is a serial number, and 'xxxxx' are 5 ASCII characters that represent a number from 1 to 60000. The gateway is configured to read these two specific UPC label formats, convert the ASCII characters into integers, and return them as DeviceNet inputs.

<u>Barcode Scanner</u>
The barcode scanner's RS232 channel is set for 9600 bps, 8 data bits, no parity, and 1 stop bit. When it reads a UPC label, it transmits the following ASCII message format. The message always begins with the ASCII STX start-of-text (0x02) character, and always ends with the ASCII ETX end-of-text (0x03) and CR carriage return (0x0D) characters. The barcode data will consist of a variable number of ASCII characters, depending upon the UPC label being scanned.

[STX] ['MODEL xxx A'] [ETX] [CR]	14 bytes ASCII data
[STX] ['SN: xxxxx'] [ETX] [CR]	12 bytes ASCII data

<u>CDN366 Gateway</u>
The CDN366 gateway needs to be configured to receive this RS232 message format. The first step is to determine the *Delimiter Mode*. The barcode scanner still transmits a variable length message, so *Length Mode* cannot be used. *Timeout Mode* may be used, but without specific timing information for the barcode scanner's RS232 channel it may be difficult to derive a suitable *Packet Timeout* value. *List Mode* is best suited for this application, because the serial message always begins and ends with the same characters. The *Serial Stream Object* can now be configured. The following shows the *Serial Stream Object* attribute settings for this application. The 3rd column lists the address string if using Set_Attribute_Single commands to write the attribute values. The last two attributes are Short_String data types.

<u>*Serial Stream Object* Configuration (Class Code 64 or 0x40)</u>

Attribute	Data	Class / Instance / Attribute / Data	Description
3. Baud Rate	9600	0x40 0x01 0x03 0x25 0x80	9600 bps
4. Data Bits	8	0x40 0x01 0x04 0x08	8 data bits
5. Parity	0	0x40 0x01 0x05 0x00	no parity
6. Stop Bits	1	0x40 0x01 0x06 0x01	1 stop bit
7. Flow Control	2	0x40 0x01 0x07 0x02	CTS / RTS
10. Delimiter Mode	1	0x40 0x01 0x0A 0x01	List Mode
11. Pre-Delimiter String	0x01 0x02	0x40 0x01 0x0B 0x01 0x02	Short_String length = 1, STX
12. Post-Delimiter String	0x02 0x03 0x0D	0x40 0x01 0x0C 0x02 0x03 0x0D	Short_String length = 2, ETX CR

The next step is to configure the CDN366 gateway to process the different label formats and convert the ASCII characters into integer numbers, to be returned to the DeviceNet scanner. With two different label formats, two Serial Receive Object Instances will be configured.

Serial Receive Object Instance 1 is configured to process the 11-character model number UPC label message packet. The *Receive Mode* is set to *Use Pre-String*, *Use Data*, and *Use Post-String* fields. The *Pre-String* attribute is set to ‘MODEL ‘, and *Post-String* is set to ‘ A’. These two strings are used to filter for the model-number message packet. The model number range is 1 to 100, so the *Data Type* is set for USINT with a *Width* of 3, and *Conversion* is set to decimal. The *Data Size* is 1 byte for USINT.

***Serial Receive Object* Instance 1 Configuration (Class Code 65 or 0x41)**

Attribute	Data	Class / Instance / Attribute / Data	Description
6. Receive Mode	7	0x41 0x01 0x06 0x07	use Data, Pre-String, Post-String fields
7. Pre-String	0x06, ‘MODEL ‘	0x41 0x01 0x07 0x06 0x4D 0x4F 0x44 0x45 0x4C 0x20	Short_String length = 6, ‘MODEL ‘
8. Post-String	0x02, ‘ A’	0x41 0x01 0x08 0x02 0x20 0x41	Short_String length = 2, ‘ A’
9. Data Type	198	0x41 0x01 0x09 0xC6	USINT (8-bit unsigned integer)
11. Width	3	0x41 0x01 0x0B 0x03	3 ASCII bytes to be converted
13. Conversion	‘D’	0x41 0x01 0x0D 0x44	ASCII bytes represent decimal number
15. Data in I/O Response	1	0x41 0x01 0x0F 0x01	Enable data in I/O response
16. Enabled	1	0x41 0x01 0x10 0x01	Instance 1 enabled

Serial Receive Object Instance 2 is configured to process the 9-character serial number UPC label message packet. The *Receive Mode* is set to *Use Pre-String* and *Use Data* fields. The *Pre-String* attribute is set to ‘SN: ‘. This string is used to filter for the serial-number message packet. The serial number range is 1 to 60000, so the *Data Type* is set for UINT with a *Width* of 5, and *Conversion* is set to decimal. The *Data Size* is 2 bytes for UINT.

***Serial Receive Object* Instance 2 Configuration (Class Code 65 or 0x41)**

Attribute	Data	Class / Instance / Attribute / Data	Description
6. Receive Mode	3	0x41 0x02 0x06 0x03	use Data, Pre-String fields
7. Pre-String	0x04, ‘SN: ‘	0x41 0x02 0x07 0x04 0x53 0x4E 0x3A 0x20	Short_String length = 4, ‘SN: ‘
9. Data Type	199	0x41 0x02 0x09 0xC7	UINT (16-bit unsigned integer)
11. Width	5	0x41 0x02 0x0B 0x05	5 ASCII bytes to be converted
13. Conversion	‘D’	0x41 0x02 0x0D 0x44	ASCII bytes represent decimal number
15. Data in I/O Response	1	0x41 0x02 0x0F 0x01	Enable data in I/O response
16. Enabled	1	0x41 0x02 0x10 0x01	Instance 2 enabled

Make sure *Serial Receive Object* instances 3-8 are disabled, since only Instances 1 and 2 are used in this application. The gateway returns 5 input bytes to the DeviceNet scanner in the I/O Response Message. The *I/O Produce Size* is 5, with the data organized as follows:

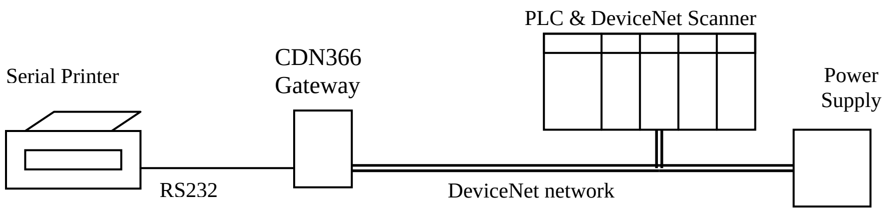
```
[ Transmit Acknowledge bits ] [ Receive Toggle bits ] [ Instance 1 USINT data ] [ Instance 2 UINT data ]
           1 byte                      1 byte                    1 byte                    2 bytes
```

The gateway is now configured to receive the barcode message packets, convert the embedded number into either an 8-bit or 16-bit unsigned integer number depending upon the scanned label type, and return it as input bytes to the DeviceNet scanner.

Receive synchronization may also be used by enabling the *Sync Enabled* attribute. The *I/O Produce Size* can be verified by reading the *Serial Stream Object’s I/O Produce Size* attribute (class 64, instance 1, attribute 20).

Example 3 – Transmitting Data

Print an ASCII string from a PLC to a serial printer, using a CDN366 gateway and a DeviceNet scanner (master). The text message string can be from 1 to 64 characters long, including any ASCII control characters. The serial printer RS232 channel is connected to a CDN366 serial channel. The CDN366 DeviceNet channel is connected to the PLC DeviceNet scanner. The DeviceNet network is powered by an external 24VDC power supply.



Serial Printer
The serial printer's RS232 channel is set for 300 bps, 7 data bits, even parity, and 2 stop bits. It uses XON / XOFF software flow control.

CDN366 Gateway
The CDN366 serial channel is configured to transmit this RS232 message format. The *Serial Stream* Object attributes are shown below for this application. The 3rd column lists the address string if using Set_Attribute_Single commands to write the attribute values.

***Serial Stream Object* Configuration (Class Code 64 or 0x40)**

Attribute	Data	Class / Instance / Attribute / Data	Description
3. Baud Rate	300	0x40 0x01 0x03 0x01 0x2C	300 bps
4. Data Bits	7	0x40 0x01 0x04 0x07	7 data bits
5. Parity	2	0x40 0x01 0x05 0x02	Even parity
6. Stop Bits	2	0x40 0x01 0x06 0x02	2 stop bit
7. Flow Control	1	0x40 0x01 0x07 0x01	XON / XOFF

The next step is to configure the CDN366 gateway to transmit data received from the DeviceNet scanner to the serial printer. To allow the printing of any text message, the gateway is configured to pass through the ASCII data bytes from the scanner to the printer. The gateway will receive a Short_String variable from the scanner. Only one *Serial Transmit Object* Instance will be configured, to process the one data variable. The *Data Type* is Short_String, with a *Data Size* of 65 (maximum text message size is 64, plus the length byte). The *Transmit Mode* is *Use Data*.

Serial Transmit Object Instance 1 Configuration (Class Code 66 or 0x42)

Attribute	Data	Class / Instance / Attribute / Data	Description
6. Transmit Mode	1	0x42 0x01 0x06 0x01	Use Data
9. Data Type	218	0x42 0x01 0x09 0xDA	Short_String
10. Data Size	65	0x42 0x01 0x0A 0x41	1 length byte, 64 data bytes
15. Data in I/O Command	1	0x42 0x01 0x0F 0x01	Enable data in I/O command

Make sure *Serial Transmit Object* instances 2-8 are disabled, since only Instance 1 is used in this application. The gateway expects to receive 67 output bytes from the DeviceNet scanner in the I/O Command Message. The *I/O Produce Size* is 67, with the data organized as follows:

```
[ Transmit Toggle bits 0000000x ] [ Receive Acknowledge bits ] [ Instance 1 Short_String printer data ]
             1 byte                          1 byte                            65 bytes
```

The gateway is now configured to receive ASCII text messages up to 64 characters in length and send them to the serial printer. The DeviceNet scanner will always send 67 outputs in the I/O Command Message, even if the text message is shorter than 64 characters. The gateway uses the Short_String length byte to determine the number of valid characters to be transmitted.

Transmit synchronization must be used by the application. The application toggles the Instance 1 *Transmit Toggle* bit in the I/O Command Message when it sends a new text message, and monitors the Instance 1 *Transmit Acknowledge* bit returned in the I/O Response Message. When the *Transmit Acknowledge* bit equals the *Transmit Toggle* bit, then the application can send the next text message.

The *I/O Consume Size* can be verified by reading the *Serial Stream Object's I/O Consume Size* attribute (class 64, instance 1, attribute 21).

Example 4 – Transmitting Delimited Data

Using the same configuration as Example 3, the CDN366 gateway is configured to print two specific text messages. For one message, the gateway converts two integer variables and inserts it into the text. The second message contains no variables, but simply prints a fixed text message. The two messages are listed below:

Message #1: 'TEMP = xxx C, xxx F', <CR>, <LF> (xxx is value, range of –50 to +400)
Message #2: 'ALARM', <CR>, <LF>

Serial Printer
The serial printer's RS232 channel is set for 300 bps, 7 data bits, even parity, and 2 stop bits. It uses XON / XOFF software flow control.

CDN366 Gateway
The CDN366 gateway is configured to transmit this RS232 message format. The *Serial Stream Object* attributes are shown below for this application. The 3rd column lists the address string if using Set_Attribute_Single commands to write the attribute values.

Serial Stream Object Configuration (Class Code 64 or 0x40)

Attribute	Data	Class / Instance / Attribute / Data	Description
3. Baud Rate	300	0x40 0x01 0x03 0x01 0x2C	300 bps
4. Data Bits	7	0x40 0x01 0x04 0x07	7 data bits
5. Parity	2	0x40 0x01 0x05 0x02	Even parity
6. Stop Bits	2	0x40 0x01 0x06 0x02	2 stop bit
7. Flow Control	1	0x40 0x01 0x07 0x01	XON / XOFF

The next step is to configure the CDN366 gateway to transmit the specific messages. Three *Serial Transmit Object* Instances are used, two for Message #1 (two variables) and one for Message #2 (one text message).

Serial Transmit Object Instance 1 is configured to transmit the first part of Message #1 (TEMP = xxx C, '). It receives an integer value from the DeviceNet scanner, converts it to 3 ASCII characters, builds a message packet, and transmits it. The *Transmit Mode* is *Use String1 Before Data*, *Use Data*, and *Use String2 After Data*. *String1* is ['TEMP = ']. String2 is [' C, ']. The *Data Type* is INT, to cover the –50 to 400 range. The *Width* is 3, and the *Conversion* is set for decimal with no leading zeros.

***Serial Transmit Object* Instance 1 Configuration (Class Code 66 or 0x42)**

Attribute	Data	Class / Instance / Attribute / Data	Description
6. Transmit Mode	19	0x42 0x01 0x06 0x13	Use Data, String1 Before, String2 After
7. String1	0x07, 'TEMP = '	0x42 0x01 0x07 0x07 0x54 0x45 0x4D 0x50 0x20 0x3D 0x20	Short String length = 7, 'TEMP = '
8. String2	0x04, ' C, '	0x42 0x01 0x08 0x04 0x20 0x43 0x2C 0x20	Short String length = 4, ' C, '
9. Data Type	195	0x42 0x01 0x09 0xC3	INT (16-bit signed integer)
11. Width	3	0x42 0x01 0x0B 0x03	convert to 3 ASCII bytes
13. Conversion	1	0x42 0x01 0x0D 0x01	represent integer in decimal
15. Data in I/O Command	1	0x42 0x01 0x0F 0x01	enable data in I/O command

Serial Transmit Object Instance 2 is configured to transmit the last part of Message #1 ('xxx F', <CR>, <LF>). It receives an integer value from the DeviceNet scanner, converts it to 3 ASCII characters, builds a message packet, and transmits it. The *Transmit Mode* is *Use Data* and *Use String1 After Data*. *String1* is [' F', <CR>, <LF>]. The *Data Type* is INT, to cover the –50 to 400 range. The *Width* is 3, and the *Conversion* is set for decimal with no leading zeros.

***Serial Transmit Object* Instance 2 Configuration (Class Code 66 or 0x42)**

Attribute	Data	Class / Instance / Attribute / Data	Description
6. Transmit Mode	9	0x42 0x02 0x06 0x09	Use Data, String1 After
7. String1	0x04 ' F' CR LF	0x42 0x02 0x07 0x04 0x20 0x46 0x0D 0x0A	Short String length = 4, ' F' <CR> <LF>
9. Data Type	195	0x42 0x02 0x09 0xC3	INT (16-bit signed integer)
11. Width	3	0x42 0x02 0x0B 0x03	convert to 3 ASCII bytes
13. Conversion	1	0x42 0x02 0x0D 0x01	represent integer in decimal
15. Data in I/O Command	1	0x42 0x02 0x0F 0x01	enable data in I/O command

Se*rial Transmit Object* Instance 3 is configured to transmit Message #2. It receives no data from the DeviceNet scanner, but is instead triggered by the *Transmit Toggle* bit. Set the *Data Type* to USINT, to minimize the *Data Size* to 1 byte, and *Width* to 1. The *Transmit Mode* is set to *Use String1 Before*. String1 is ['ALARM', CR, LF].

***Serial Transmit Object* Instance 3 Configuration (Class Code 66 or 0x42)**

Attribute	Data	Class / Instance / Attribute / Data	Description
6. Transmit Mode	2	0x42 0x03 0x06 0x02	Use String1 Before
7. String1	0x07, 'ALARM', CR, LF	0x42 0x03 0x07 0x41 0x4C 0x41 0x52 0x4D 0x0D 0x0A	Short String length = 7, 'ALARM' <CR> <LF>
9. Data Type	198	0x42 0x03 0x09 0xC6	USINT (8-bit unsigned integer)
11. Width	1	0x42 0x03 0x0B 0x01	convert to 1 ASCII bytes
15. Data in I/O Command	1	0x42 0x03 0x0F 0x01	Enable data in I/O command

Make sure the Serial Transmit Object instances 4-8 are disabled, since only Instances 1-3 are used in this application. The gateway expects to receive 7 output bytes from the DeviceNet scanner in the I/O Command Message. The I/O Produce Size is 7, with the data organized as follows:

```
[ TX Toggle bits 000000xx ] [ RX Acknowledge bits ] [ Inst 1 data ] [ Inst 2 data ] [ Inst 3 data ]
          1 byte                      1 byte              2 bytes         2 bytes         1 byte
```

The application should send Instance 1 and Instance 2 data bytes at the same time, so that the values can be converted and transmitted sequentially to build Message #1. Instance 1 will build its message packet first and load it into the transmit buffer. Instance 2 will build its message next and load it into the transmit buffer. The result is the transmission of the entire Message #1 string, complete with temperature values in C and F. Because the gateway does not support a NULL data byte, the scanner must still send a data value to Instance 3. The Instance does not use the byte, but instead is triggered by its *Transmit Toggle* bit to send Message #2.

Transmit synchronization must be used. The application toggles Instance 1 and 2 *Transmit Toggle* bits in the I/O Command Message when it sends new temperature values, and monitors Instance 1 and 2 *Transmit Acknowledge* bits to tell when the message has been sent. The application toggles Instance 3 *Transmit Toggle* bit to transmit Message #2, and monitors the Instance 2 *Transmit Acknowledge* bit to tell when the message has been sent.

Chapter 8 – Troubleshooting

Problem	Possible Cause
DeviceNet Configuration Program does not recognize Gateway.	• Register Gateway EDS file with Configuration Program.
DeviceNet Configuration Program does not recognize Gateway after loading EDS file.	• Check Major and Minor Revisions for Gateway and EDS file, to see if you have correct EDS file for your Gateway's firmware version.
Gateway does not appear on DeviceNet network.	• Check wiring and cable connections. • Check DeviceNet power supply voltage. • Make sure Gateway baud rate matches network baud rate. • Verify Gateway baud rate is set from rotary switches or retentive memory value. • Make sure Gateway MAC ID is not used by another device.
After setting Gateway MAC ID, DeviceNet Master does not recognize Gateway.	• Disconnect Gateway from network before changing MAC ID. • Make sure Gateway MAC ID is not used by another device. • Verify Gateway MAC ID is set from rotary switches or retentive memory value. • Verify DeviceNet baud rate.
NET LED is flashing red.	• Gateway is removed from DeviceNet Master scanlist or network. Power cycle Gateway to reset.
NET LED is solid red.	• Make sure Gateway MAC ID is not used by another device. Possible DeviceNet network failure.
NET LED is off.	• Check wiring and cable connections. • Check DeviceNet power supply voltage. • Make sure Gateway baud rate matches network baud rate. • Verify Gateway baud rate is set from rotary switches or retentive memory value.
MOD LED is flashing or solid red.	• Gateway has failed. Cycle power to reset. Replace Gateway if necessary.
RX LED does not flash green when data is sent to the Gateway.	• If Sync Enabled, make sure Receive Toggle and Receive Acknowledge bits are being toggled. If application does not toggle Receive Acknowledge, Gateway will not receive data. Verify data is being received in Receive Data. • If Sync Enabled is disabled, verify data is being received in Receive Data. • Verify source device is transmitting data to Gateway.
RX LED is solid red after Gateway receives data.	• Check Serial Status for RX buffer Overflow or Parity Error. Reset Gateway or clear Serial Status error bit if necessary. • Make sure parity is set to match transmitting device.
TX LED is solid red after receiving data from DeviceNet Master.	• Ceck Serial Status for TX buffer Overflow or Parity Error. Reset Gateway or clear Serial Status error bit if necessary. • Make sure parity is set to match receiving device configuration.
TX LED does not flash green when Gateway should be transmitting data.	• Make sure Transmit Toggle is being toggled. If applicatoin does not toggle Transmit Toggle, Gateway will not transmit data. • Verify data is being saved in Transmit Data.
1747-SDN Scanner displays error code 77.	• Gateway I/O Produce Size and/or I/O Consume Size value do not match 1747-SDN I/O Rx/Tx settings.

Appendix A – Product Specifications

DeviceNet Interface

Power Requirements:	11 - 28 Vdc @ 50 mA
Loss of Ground:	Yes
Reverse Polarity:	-30 Vdc
Signal Levels:	ISO11898

Serial Channel

Isolation:	500 Volts
ESD Protection:	+/- 10 kV
Overload Protection:	+/- 30 Volts
Short Circuit:	Indefinite
RS232 Output Levels:	+/- 7.9 Volts (unloaded, typical)

Environmental

Operating Temperature:	0^o C to 70^o C
Storage Temperature:	-25^o C to 85^o C
Size (inches):	3.25 x 2.37 x 1.08
Mounting (inches)	0.5 tabs, 3/16 diameter mounting holes
PCB Encapsulation:	RTV Silicon Compound

Appendix B – DeviceNet Template

Class	*Instance*	*Attribute*	*Default*	*Setting*	*Unit*	*Comments*

Appendix C – ASCII Character Codes

Non-Printable Characters					Printable Characters								
Hex	**Dec**	**Char**	**Name**	**Kybd**	**Hex**	**Dec**	**Char**	**Hex**	**Dec**	**Char**	**Hex**	**Dec**	**Char**
0x00	0	NUL	Null	Ctrl @	0x20	32	Space	0x40	64	@	0x60	96	`
0x01	1	SOH	Start of heading	Ctrl A	0x21	33	!	0x41	65	A	0x61	97	a
0x02	2	STX	Start of text	Ctrl B	0x22	34	"	0x42	66	B	0x62	98	b
0x03	3	ETX	End of text	Ctrl C	0x23	35	#	0x43	67	C	0x63	99	c
0x04	4	EOT	End of transmit	Ctrl D	0x24	36	$	0x44	68	D	0x64	100	d
0x05	5	ENQ	Enquiry	Ctrl E	0x25	37	%	0x45	69	E	0x65	101	e
0x06	6	ACK	Acknowledge	Ctrl F	0x26	38	&	0x46	70	F	0x66	102	f
0x07	7	BEL	Bell	Ctrl G	0x27	39	'	0x47	71	G	0x67	103	g
0x08	8	BS	Backspace	Ctrl H	0x28	40	(	0x48	72	H	0x68	104	h
0x09	9	HT	Horizontal tab	Ctrl I	0x29	41	)	0x49	73	I	0x69	105	i
0x0A	10	LF	Line feed	Ctrl J	0x2A	42	*	0x4A	74	J	0x6A	106	j
0x0B	11	VT	Vertical tab	Ctrl K	0x2B	43	+	0x4B	75	K	0x6B	107	k
0x0C	12	FF	Form feed	Ctrl L	0x2C	44	,	0x4C	76	L	0x6C	108	l
0x0D	13	CR	Carriage return	Ctrl M	0x2D	45	-	0x4D	77	M	0x6D	109	m
0x0E	14	SO	Shift out	Ctrl N	0x2E	46	.	0x4E	78	N	0x6E	110	n
0x0F	15	SI	Shift in	Ctrl O	0x2F	47	/	0x4F	79	O	0x6F	111	o
0x10	16	DLE	Data line escape	Ctrl P	0x30	48	0	0x50	80	P	0x70	112	p
0x11	17	DC1	Device control 1	Ctrl Q	0x31	49	1	0x51	81	Q	0x71	113	q
0x12	18	DC2	Device control 2	Ctrl R	0x32	50	2	0x52	82	R	0x72	114	r
0x13	19	DC3	Device control 3	Ctrl S	0x33	51	3	0x53	83	S	0x73	115	s
0x14	20	DC4	Device control 4	Ctrl T	0x34	52	4	0x54	84	T	0x74	116	t
0x15	21	NAK	Negative acknowledge	Ctrl U	0x35	53	5	0x55	85	U	0x75	117	u
0x16	22	SYN	Synchronous idle	Ctrl V	0x36	53	6	0x56	86	V	0x76	118	v
0x17	23	ETB	End of transmit block	Ctrl W	0x37	55	7	0x57	87	W	0x77	119	w
0x18	24	CAN	Cancel	Ctrl X	0x38	56	8	0x58	88	X	0x78	120	x
0x19	25	EM	End of medium	Ctrl Y	0x39	57	9	0x59	89	Y	0x79	121	y
0x1A	26	SUB	Substitute	Ctrl Z	0x3A	58	:	0x5A	90	Z	0x7A	122	z
0x1B	27	ESC	Escape	Ctrl [	0x3B	59	;	0x5B	91	[	0x7B	123	{
0x1C	28	FS	File separator	Ctrl \	0x3C	60	<	0x5C	92	\	0x7C	124	|
0x1D	29	GS	Group separator	Ctrl]	0x3D	61	=	0x5D	93	]	0x7D	125	}
0x1E	30	RS	Record separator	Ctrl ^	0x3E	62	>	0x5E	94	^	0x7E	126	~
0x1F	31	US	Unit separator	Ctrl _	0x3F	63	?	0x5F	95	_	0x7F	127	DEL