Huron Net Works, Inc.

DN-SLIP

DeviceNet Serial Link Interface Protocol

Functional Specification

Publication #2200092 Revision History

Revision	Date	Description
0.1	4/26/02	Original pre-release
0.2	5/1/02	Add Serial Link Object
0.3	5/2/02	Add Diagnostic Modes
0.4	5/17/02	Implementation Revisions, Wait for baudrate and MACId Proposal
0.5	6/12/02	Serial Link Object Revisions
0.6	6/28/02	MacId and Baudrate Override

Preface

This specification is inspired by the referenced works, and provides a straightforward mechanism for the encapsulation and transmission of DeviceNet “datagrams” over serial lines. Although the term “datagram” is not referenced in the DeviceNet specification it fits very well.

> “If I have been able to see further, it was only because I stood on the shoulders of giants.”
>
> *Sir Isaac Newton*

Notation

Object Instance Attribute Addresses

are represented as a triple of numbers enclosed in curly braces. For example the notation {3,1,1} represents the DeviceNet Class, Class #3, Instance #1 of the DeviceNet Class, and Attribute #1 of Instance #1 of the DeviceNet Class. This particular reference refers to the DeviceNet MacId.

References

1. DeviceNet Specification, Volumes I and II, Release 2.0, Errata 5
2. Romkey, J.,STD0047 (RFC 1055), A NONSTANDARD FOR TRANSMISSION OF IP DATAGRAMS OVER SERIAL LINES: SLIP, JUNE 1988.

1 Introduction

1.1 Purpose

The purpose of this specification is to define methods and protocol for adding a Serial Link to a DeviceNet product. It covers the low level framing protocol and how to encapsulate DeviceNet messages so that they may be passed to and from a device at the other end of the Serial Link.

There are many potential application specifications for how messages are handled. The initial application specification allows I/O messages to pass through the device in a transparent fashion. Explicit messages on the Serial Link are processed locally within the device and are not passed onto the DeviceNet network. The acronym for this application is LETIO which stands for Local Explicit and Transparent I/O.

1.2 Environment

The DeviceNet Serial Link Interface Protocol applies to a product that has both a DeviceNet/CAN network attachment and an RS-232/RS-485 Serial UART attachment. Some assumptions about these attachments will drive some of the aspects of this specification.

1.2.1 Serial Link

The serial link physical layer will attach to an Asynchronous UART. The most likely configuration of the UART parameters will be:

Parameter	Value
Parity	None
Data Bits	8
Stop Bits	1
Baudrate	9600

Other baudrates and serial framing combinations are possible and the particular values will be encoded as attributes of a Serial Link Object.

The serial link is assumed to be point to point. So the concepts of source and destination addresses are not relevant. Since that is the case, the non DeviceNet end of the serial link, may assign any value to any MacId address bits it may need to use in constructing a DeviceNet identifier . If a device knows the value of the MacId address bits for a DeviceNet connection, then that value can be used. If a device does not know what value of MacId address bits to use, all zeros is the preferred, but not required choice.

1.2.2 DeviceNet Link

The DeviceNet Link will attach to a CAN Controller. It will be assumed that all three baudrates (500 kbps,250 kbps,125 kbps) will be supported. It is assumed that the DeviceNet side has a method of determining MacId and baudrate independently of the serial link. The serial link may have the ability to change or override the choice of baudrate and MacId for the DeviceNet connection. If this is allowed then the determination of MacId and baudrate at Power Up/Reset will occur as follows:

If the device has switches, and the baudrate switches are in the set {00, 01, 10} corresponding to the standard baudrates: 125kbps, 250kbps, and 500kbps then baudrate and MacId are determined as follows:

1. Get baudrate and MacId from the switches
2. Execute the Network Access State Machine
3. Go Online
4. Set_Attribute_Single to baudrate or MacId returns an error

If the device has no switches or has switches, and the baudrate switches are in the set {11} corresponding to an undefined baudrate value, then baudrate and MacId are determined as follows:

1. Get baudrate and MacId from Non-Volitile memory or, if Non-Volitile memory is invalid use the default baudrate of125kbps, and default MacId of 63
2. Execute the Network Access State Machine
3. Go Online
4. Set_Attribute_Single to baudrate or MACId overrides the current settings

In neither case does the device have to wait for baudrate or MACId.

1.2.2.1 MacId and Baudrate Override

There will be a configuration parameter, with permission bits, which say if either the DeviceNet side or the Serial Link side is allowed to override the MacId and Baudrate. If there are hardware switches then the switches must be in the {11xx xxxx} undefined or soft settable state. It the switches are in the correct state and the permission bit is a 1 then a Set Attribute Single to the MacId {3,1,1} or the Baudrate {3,1,2} will change the respective value. When the MacId is changed the device will execute the Network Access State Machine. That means the two DUP MAC Check Messages are sent out with the new MacId. When the Baudrate is changed, it does not take effect until there is a RESET to the Identity Object or a power cycle.

2 Basic Serial Link Framing

2.1 Begin/End Frame Delimiter

Each serial link frame ends with a framing delimiter, 0xC0. A serial link transmitter may send multiple framing delimiters without causing a serial link receiver any upset. A serial link transmitter can begin a frame with an End Delimiter to clear out any characters, which may be present in a serial link receiver. A serial link receiver will discard multiple consecutive framing delimiters except the last one. In the following paragraphs the framing delimiter will be referred to as just the End Delimiter.

2.2 Escape Sequence

If a data byte between the framing delimiters has the value 0xC0 then a serial link transmitter will send the two character sequence 0xDB, 0xDC. A serial link receiver will convert the sequence 0xDB, 0xDC back to a data value of 0xC0. If a data byte between the framing characters has the value 0xDB then a serial link transmitter will send the two character sequence 0xDB, 0xDD. Escape sequences do not affect checksum or CRC algorithms. The checksum or CRC is computed on the actual data values in the frame and not on the escape sequence values.

2.3 Transmitter Behavior

A serial link transmitter should begin each frame with a framing character. It should end each frame with a framing character. There should be, as a parameter or a pre-defined constant, a maximum number of characters that can occur between the framing characters. It should be a goal of the application firmware to have as an upper bound on the transmit time of:

$$4*\frac{1}{9600\frac{bits}{\sec}}*10\frac{bits}{char}*N\frac{char}{frame}=UB\frac{\sec}{frame}$$

A serial link transmitter may join together multiple frames without sending multiple framing characters. That is, the End Delimiter of one frame can be the Begin Delimiter of the next frame, or conversely the Begin Delimiter of a frame can be the End Delimiter of the previous frame.

2.4 Receiver Behavior

2.4.1 Normal Behavior

A serial link receiver should ignore all incoming characters until it sees a framing character. It should ignore multiple consecutive framing characters except the

last one. The same parameter or pre-defined constant, which specifies a maximum number of characters between the framing characters for the serial link transmitter, also applies to the serial link receiver. This means that if this limit is exceeded the receiver can throw away the long frame and wait for the next End Delimiter.
The escape sequences are processed as follows:

$$0xDB, 0xDC \rightarrow 0xC0$$
$$0xDB, 0xDD \rightarrow 0xDB$$

In the case where the character following the escape character (0xDB) is not one of the legal successor characters a protocol violation has occurred. In this case the escape character is ignored and the character is accepted literally. In some cases a length check or checksum calculation can detect this kind of error. Any invalid frame may be discarded and ignored. No response to an invalid frame is required.

Once a valid frame has been received it should be a goal of the application firmware to begin a required response within ten (10) character times or:

$$10 * 10\frac{bits}{char} * \frac{1}{9600\frac{bits}{\sec}} \approx 10\ m\sec.$$

These conditions on transmission time and turnaround time will allow and upper bound on the total transaction time to be calculated.

2.4.2 Receiver Error Behavior

All error conditions detected by the Serial Link Receiver including:

- Character framing and/or parity errors
- Checksum errors
- Too many characters between framing delimiters
- I/O message longer than produced connection size
- Invalid Identifiers

Are handled by ignoring the entire frame and transmitting no response.

3 DeviceNet Message Encapsulation

3.1 Unfragmented Messages

The essential parts of an unfragmented DeviceNet message are the 11-bit CAN identifier and the [0..64] bit data field. In a CAN frame the length of the valid data is given explicitly. In the Serial Link frame the length will be determined implicitly because, in most cases, in the transfer of I/O messages the same number of bytes are always transferred, and this is the process we wish to optimize.

To frame a DeviceNet message for the Serial Link we concatenate the 11-bit identifier field, right justified and zero filled, with the zero to eight-byte data field followed by a one byte 2's complement checksum of the identifier and the data field. Any numeric quantity whose representation exceeds 8 bits will be sent over the Serial Link in little-endian form.

Example 1: Over the Serial Link a message to change the Poll Response Data from its current value to <0x00, 0x00> would look like:

C0	DB	DC	03	00	00	3D	C0
End	ID.low = C0		ID.high	Data	Data	CkSum	End

The ID field of 0x3C0 is the CAN identifier for the DeviceNet message: "Poll Response from node 0". The use of node 0 as the source of a DeviceNet message is actually a polite fiction since the real node zero either does not exist or is on the DeviceNet network. On the Serial Link, the sender of a Poll Response message may use any value for the Source MACId bits. On the Serial Link the receiver of a Poll Response message will ignore these bits. The new data of <00 00> will be returned over DeviceNet in a Poll Response Message which is triggered by the consumption of a Poll Request Message. It is a coincidence that the DeviceNet/CAN Identifier chosen for this example happens to be a value that requires the use of an escape sequence. The Poll Response Identifier of 0x3C0 decodes as a Group 1 Message, Message Id 15, Source MACId 0. Since we could use any source MACId, we could save a byte in transmitting this message by using Identifier 0x3C1. If we did this, the above message would look like:

C0	C1	03	00	00	3C	C0	
End	ID = 3C1		Data	Data	CkSum	End	

Over the Serial Link we do not care about the value of any MACId bits. If the message comes to us over the Serial Link we must be the correct destination. When we respond we always use our current valid MACId. The client endpoint of the DN-SLIP connection could use this value of MACId in subsequent messages, but this is not required.

3.2 Fragmented Messages

The Serial Link does not have the same message length limitations as the DeviceNet/CAN side. For this version of the DN-SLIP application protocol the data field of a frame is from zero to 256 bytes in length. With the addition of a two-byte identifier and a one-byte checksum this means there can be up to 259 characters between the End Delimiters. The character stuffing escape sequence does not affect this calculation but it does affect the time to transmit or receive a frame. In this version of the application protocol, messages are not passed directly from DeviceNet to the Serial Link, so we do not need to take account of fragmentation.

Example 2: The current data used to construct a Poll Response on the DeviceNet Link is the following ten byte string:

<01 22 03 44 05 66 07 88 09 AA>

and we want to change it the following ten byte string:

<11 02 33 04 55 06 77 08 99 0A>

This would be accomplished with the following message over the Serial Link:

C0	DB	DC	03	11	02	33	04
End	ID.low = C0		ID.high	Data	Data	Data	Data

55	06	77	08	99	0A	76	C0
Data	Data	Data	Data	Data	Data	CkSum	End

Since message length over the Serial Link is determined implicitly we can send a message with a length longer than eight bytes, which would have required fragmentation on the DeviceNet side.

4 Message Specific Behavior

4.1 Transparent I/O Message Behavior

4.1.1 Poll Response Messages – Identifiers in the range {[3C0]..[3FF]}

A Poll Response Message has [0,15,xx] for an identifier. This decodes as Group 1, Message ID 15, and any Source MACId. When the Serial Link Receiver receives this message it performs an atomic update on an area of memory that is used to construct the response to a Master's Poll Request [2,xx,5], which decodes as Group 2, any destination MACId, Message ID 5. The Serial Link Transmitter should never transmit a message with an Identifier in the Poll Response Message range. In addition to the atomic update of the Poll Response Data, the Serial Link Transmitter will respond with a message containing the Poll Request Data.

4.1.2 Poll Request Messages – Identifiers in the set { [405], [40D], …[5FF]}

A Poll Request Message [2,xx,5] should never be received by the Serial Link Receiver. It is the response to the consumption of the Poll Response Message by the Serial Link Receiver. The data field of a Poll Request Message is an atomic snapshot of the most recent incoming Poll Request Data from the DeviceNet port. The correct MacId for the device will be used to construct the identifierin the serial frame.

4.1.3 COS/Cyclic I/O Messages – Identifiers in the range {[340]..[37F]}

A COS/Cyclic I/O Message has [0,13,xx] for an Identifier. This decodes as Group 1, Message ID 13, and any source MacId. When the Serial Link Receiver receives this message it performs an atomic update on an area of memory that is used to construct the COS/Cyclic message. This will probably trigger a change of state production on the DeviceNet link. The Serial Link Transmitter should never transmit a message in the COS/Cyclic I/O message range. In addition to the atomic update of the COS/Cyclic response data, the Serial Link Transmitter will respond with a Change of State Acknowledge Message [2,xx,2] that contains no data. This response will ignore the state of the ACK Suppress bit, which governs the behavior on the DeviceNet side.

4.1.4 COS/Cyclic Acknowledge Messages – Identifiers in the set {[402], [40A], …, [5FA]}

The Serial Link Receiver should never receive a COS/Cyclic Acknowledge Message. It is the response to the consumption of the COS/Cyclic I/O Message. The data field of this message is empty. The correct MACId for the device will be used to construct the identifier in the serial frame.

4.1.5 Bit Strobe Response/Request

All message identifiers associated with the Bit Strobe Connection will be ignored by DN-SLIP.

4.1.6 Multicast Poll Response/Request

All message identifiers associated with the Multicast Poll Connection will be ignored by DN-SLIP.

4.2 Explicit Message Behavior

4.2.1 Master's Explicit Request Messages – Identifiers in the set {[404], [40C], …, [5FC]}

A Master's Explicit Request [2,xx,4] will behave as if the Serial Link was the same as the DeviceNet Link. That is the request will be processed locally by the device as if it had come in over DeviceNet. This means that the device at the other end of the Serial Link has an ability similar to the DeviceNet Master. It can use services like Get_Attribute_Single and Set_Attribute_Single to read and write attributes(data) in the device. The rules on which attributes, of which instances, of which objects are visible and/or settable will be **different** between the Serial Link and the DeviceNet Link. The data field follows the format of the Explicit Message on the DeviceNet side. The most likely format for the message body will be 8-bit class and 8-bit instance. The Frag bit in the Explicit Message Header should always be set to zero since fragmentation is not necessary on the Serial Link.

4.2.2 Slave's Explicit Response Messages – Identifiers in the set {[403], [40B], …[5FB]}

The Slave's Explicit Response [2,xx,3], which decodes as Group 2, my MACId, and Message Id 3 will be sent in response to a Master's explicit request. The format of the data will be the same as for the DeviceNet Explicit Response including the Error Response. The Frag bit in the Explicit Message Header will always be set to zero since fragmentation is not necessary on the Serial Link.

4.3 Other Message Identifiers

Other message identifiers, which are part of the DeviceNet/CAN Protocol in the range of [0..2047], not specifically defined above, are undefined in DN-SLIP. Identifiers in the range 2048-65535 are reserved for future applications and are undefined..

4.3.1 Undefined Identifier

The reception of a frame with an undefined identifier causes the frame to be discarded and no response to be sent.

5 DN-SLIP Client Description

5.1 Interrogating a DN-SLIP Server

The client side of a DN-SLIP connection, over a Serial Link, can use the pre-defined and always available Serial Link connection. This connection will not be represented by an instance of the connection object. It is not important to capture the behavior of individual connections over the Serial Link. The pre-defined Serial Link Connection will be used in conjunction with the Serial Link Object, which is patterned after the DeviceNet Object.

5.1.1 Explicit Messages

When using Explicit Message Identifiers, the client, sends Explicit Requests [2,xx,4] and receives Explicit Responses [2,xx,3] over the Serial Link.

5.1.2 Poll I/O Messages

With respect to Poll I/O Messages the client sends a Poll Response Message [0,15,xx] to update the data in the DeviceNet server. The DeviceNet server responds with a Poll Request Message [2,xx,5] to inform the client of the most recent Poll Request data which has arrived from the DeviceNet port.

5.1.3 COS/Cyclic Messages

The client sends COS/Cyclic Messages [1,D,xx] to update the data in the DeviceNet server. The DeviceNet Server responds with a COS Ack[2,xx,2] and no data.

6 Serial Link Object

Class Code: 0x6A

This object provides the support for a serial link connection, attached to an asynchronous UART device. There should be one instance of this object for each physical or software simulated UART.

6.1 Class Attributes

Attribute ID	Need in Implementation	Access Rule	Name	DeviceNet Data Type	Description of Attribute
1	Conditional	Get	Revision	UINT	Revision of this object Current Value = 0x0001
2	Conditional	Get	Max Instance	UINT	Maximum instance number
3-7	Optional	These attributes are optional and described in Volume II chapter 5 of the DeviceNet Specification			

6.2 Instance Attributes

Attribute ID	Need in Implementation	Access Rule	Name	DeviceNet Data Type	Description of Attribute
1	Required	Set	Baudrate	UINT	Nominal Baudrate in bits per second. Default is: 9534 = 0x253E
2	Required	Set	Mode	USINT	‘A’ – Asyncronous (Default) ‘S’ – Syncronous
3	Required	Set	Parity	USINT	‘N’ – No Parity (Default) ‘E’ – Even Parity ‘O’ – Odd Parity ‘0’ – Zero Stick Parity ‘1’ – One Stick Parity
4	Required	Set	Data Bits	USINT	‘5’ – 5-bit ‘6’ – 6-bit ‘7’ – 7-bit ‘8’ – 8-bit (Default)
5	Required	Set	Stop Bits	USINT	‘1’ – One Stop Bit (Default) ‘2’ – Two Stop Bits ‘H’ – One and ½ Stop Bits
6	Required	Set	Check Sum	USINT	‘0’ – 2’s Complement (Default) ‘1’ – Straight Sum
7	Required	Set	Diagnostic Action	USINT	‘0’ – No Diagnostic Action ‘1’ – Echo ‘2’ – Transmit Character ‘3’ – Transmit Frame
8	Required	Set	Diagnostic Character	USINT	Any value in the range [0..255]
9	Required	Set	Link Timer	UINT	Any value in the range [0..65535] in milliseconds
10	Optional	Set	Rx Framing Error	USINT	[0..255]
11	Optional	Set	Rx Data Overrun	USINT	[0..255]
12	Optional	Set	Tx Data Overrun	USINT	[0..255]
13	Optional	Set	Rx_Proc Overun	USINT	[0..255]
14	Optional	Set	Rx Buffer	ARRAY	13 bytes

15	Optional	Set	Rx Buffer Length	USINT	[0..255]
16	Optional	Set	Diag 0	USINT	[0..255]
17	Optional	Set	Diag 1	USINT	[0..255]
18	Optional	Set	Diag 2	USINT	[0..255]

Each of the instance attributes in the range[1..9] is required in an implementation. A product is not required to support all possible values for an attribute. If only one alternative is supported then the access rule may be restricted to Get. If none of the attributes are settable, then they just document the implementation. It is also possible that different rules may apply for Explicit Requests from the Serial Link than Explicti Requests from the DeviceNet Link.

6.3 Common Services

The Serial Link Object provides the following Common Services

Service Code	**Need in Implementation**		**Service Name**	**Description of Service**
	Class	Instance		
0x0E	Conditional	Required	Get_Attribute_Single	Returns the content of the specified Attribute
0x10	No	Conditional	Set_Attributte_Single	Modifies the value of an Attribute

6.4 Diagnostic Action Behavior

Each of the following diagnostic action modes is entered by setting the value of the Diagnostic Action attribute to a particular non-zero value with an Explicit Message.

6.4.1.1 Diagnostic Action 1: Echo

In this state all characters consumed by the Serial Link Receiver are echoed to the Serial Link Transmitter as quickly as possible. The Serial Link Object will stay in this state until the Diagnostic Action attribute changed.

6.4.1.2 Diagnostic Action 2: Transmit Diagnostic Character

In this state, the Serial Link Transmitter will send the Diagnostic Character continuously at a rate determined by the Serial Link Timer which is expressed in units of milliseconds. An End Delimiter (0xC0) will stop the continuous transmission. While stopped, any other character will caus the continuous transmission to restart.

6.4.1.3 Disgnostic Action 3: Transmit Frame

In this state, the Seria Link Transmitter will send the current frame in response to any single character.

Appendix A

Examples

1. Explicit Request to Get Vendor Id which is {1,1,1} or Class 1, Instance 1, Attribute 1. SEND

C0	04	04	00	0E	01	01	01
End	Id.low	Id.High	Hdr	Service	Class	Inst	Attr
E7	C0						
CkSum	End						

2. Explicit Response to Get Vendor Id, Assume Vendor ID = 0x0014 which is the Huron Net Works Vendor ID. Assume furthur that MACId od the DeviceNet Server is 42 or 0x2A. RECEIVE

C0	53	05	00	8E	14	00	06
End	Id.low	Id.High	Hdr	Service	VId.low	VId.high	CkSum
C0							
End							

3. Update five bytes of Poll Response Data. Data is <00 0F FA C6 55> SEND

C0	05	04	00	0F	FA	C6	55
End	Id.low	Id.High	Data	Data	Data	Data	Data
D3	C0						
CkSum	End						

4. Get four bytes of Poll Request Data. Data is <00 FF FF 00>. RECEIVE

C0	EA	03	00	FF	FF	00	15
End	Id.low	Id.High	Hdr	Service	VId.low	VId.high	CkSum
C0							
End							

From the client end point in the outgoing messages the MACId bits are don't care and always sent as zeros. In the incoming messages the server always supplies its correct MACId. In this fashion a change of MACId could be detected.