Table of Contents

Introduction

Chapter Overview

This chapter serves as an introduction to the DIP065.

Introduction

The following serial protocol has been developed to support the DIP RS232 to CAN network adapter. This protocol will be considered proprietary to DIP. The protocol will be supported on the DIP065 CAN based microcontroller hardware adapter (CAN Serial Interface Module) using a full duplex serial RS232 electrical interface.

The CAN protocol specifications are specifically not defined. It is the intent of the DIP065 module to allow various CAN protocols to be processed using software supplied on the controlling RS232 device (HOST system). The DIP065 will pass messages through with no interpretation of the data.

Data transferred on the CAN network consist of variable length packets. The maximum packet length is 8 data bytes + 2 control bytes. Since the DIP065 module is to pass data through with no attempt to interpret the CAN message protocol the serial protocol will simply prepend a suitable STX character and append a checksum/ETX character to the CAN packets.

The DIP065 module is not intended to be a control interface, but to instead provide a method to monitor and configure a CAN network or individual CAN device.

Hardware Requirements

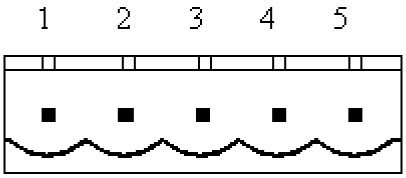
Chapter Overview

This chapter will address the DIP065 Hardware Requirements.

DeviceNet Terminal Block Connector

The DIP065 serial adapter connects to DeviceNet using the following pin description. Notice that Pin 1 and Pin 5 are **OUTPUT** pins.

⌦ Pin 1 is the leftmost pin on the unit.



Pin	Description
1	Common Out (Bus -)
2	CAN Low
3	Shield
4	CAN High
5	+15 VDC Out (Bus +)

Power Connections

The DIP065 may be powered either from a local power source (9-12 VDC 500mA) or from a 9 volt battery. The unit is able to supply +15 VDC at 100 mA using a rechargeable NiCd 150mAh for approximately one hour.

Chapter 3

Functional Requirements

Chapter Overview

This chapter will address the DIP065 Functional Requirements.

RS232 Serial Channel

The purpose of the DIP065 Serial protocol is to allow an RS232 serial channel to gain access to a CAN based control network. The RS232 channel will typically be (but in no way limited to) an IBM PC compatible personal computer (HOST system). Electrical power for the RS232 transceivers will be derived from the DSR/DTR signal pair.

Pin	Description
2	RX (Receive)*
3	TX (Transmit)*
4	DTR (Data Ready)
5	GND (Ground)*
7	RTS (Ready to Send)

⌦ These are the minimum required signals needed to operate the serial interface.

Data Rates

The RS232 serial interface will operate at 9600 baud, 1 stop bit, no parity. The CAN network operating rate will be determined by the configuration registers as set by the HOST software.

Data Buffering

CAN data rates vary from very low (less than 9600 baud) to very high (1 megabaud). The DIP065 module will allow for 176 data bytes to be buffered in a receive FIFO structure for data packets received from the CAN network pending transmission to the RS232 channel.

Due to the typically much higher data rate for the CAN network there is little need to provide RS232 to CAN data buffering. The only time the RS232 channel can operate faster than the CAN network is when the network itself is heavily loaded or a fault condition has occurred. In both cases it is considered reasonable that the HOST software will want to take corrective actions so message buffering in fact would prove a hindrance.

Data Flow

Information flowing between the DIP065 and the HOST are considered to be either DATA packets (transferred to or from the CAN channel) and CONFIGURATION packets (information used to control the operation of the DIP065 module).

Data Packets

Packets of information sent from the HOST to the CAN network will be received and buffered in an internal transmit buffer. Upon verification of message checksums and data format (ETX) the data will be transferred to the CAN transmit buffer and then the appropriate transmit control signals will be generated. At this point the transmit buffer will be considered empty and another transmit request can be accepted.

Following successful completion of the CAN transmission an ACKNOWLEDGE packet will be returned to the HOST. If errors occur an error packet will be returned to the HOST.

Packets of information received on the CAN network which are accepted by the controller will be transferred to a receive FIFO. A background handler will automatically transfer data from the receive FIFO to the RS232 serial channel, encapsulated with STX and CHECKSUM, ETX bytes.

When the HOST responds with an acknowledge the DIP065 will either retransmit the packet (error condition) or proceed with the next available packet if one is available. If the HOST does not respond the receive FIFO will eventually overflow. The DIP065 may be configured to either discard subsequent CAN packets or to throw away the oldest packets.

Configuration Packets

Configuration packets must also be acknowledged by the receiver before the transmitter is free to send further packets.

If the HOST sends a request to write configuration information the DIP065 will process the request and then return either an acceptance or failure acknowledgement.

If the HOST sends a request to read configuration information the DIP065 will process the request and return either a failure acknowledgement or the requested information. No response is expected from the HOST.

The DIP065 module will never send an unsolicited configuration packet.

Flow Control

Packets are defined as either DATA packets to/from the CAN interface or CONFIGURATION packets which transfer status and control information.

Packet flow is controlled by the receiver generating an ACKNOWLEDGE after receiving each packet. The transmitter will not initiate another transmission of the same type until an acknowledge has been received.

In cases where the receive FIFO is not empty (DIP065 sending transactions to HOST) the HOST may force a pause by not responding to a transmission. During the pause the DIP065 will continue to process configuration packets sent by the HOST.

Configuration Control

Two levels of configuration control are required. The CAN control subsystem has specific control registers to determine transmission rates and address filtering. In addition, configuration control is provided to determine the RS232 packet management.

The CAN packet header uses 4 bits to define the packet length, allowing for potentially 16 different message lengths (0-15). Only packet lengths of 0-8 are acceptable, leaving 7 undefined lengths which can be used to transfer configuration information. The following configuration transfers will be supported:

Code 9 allows the RS232 channel to access the CAN interface set up registers.

```
OBJ         = 0         ; RESERVED
RTRLEN      = 0x09      ; Write configuration
DATA                    ; ACCEPTANCE code
DATA+1                  ; MASK code
DATA+2                  ; TIM0 code
DATA+3                  ; TIM1 code

OBJ         = 0         ; RESERVED
RTRLEN      = 0x19      ; Write configuration
DATA                    ; ACCEPTANCE code
DATA+1                  ; MASK code
DATA+2                  ; TIM0 code
DATA+3                  ; TIM1 code
```

Code 10 allows the RS232 channel to access the CAN interface control/status registers.

```
OBJ         = 0      ; RESERVED
RTRLEN      = 0x0A   ; Write configuration or
DATA                 ; CAN_CONTROL

OBJ         = 0
RTRLEN      = 0x1A   ; Read configuration
DATA                 ; CAN_STATUS
```

Code 11 allows the RS232 channel to set the XHold mode and the DiscardMode registers and to reset the device. The first byte contains the mode byte, shown below. The second byte must be 0.

```
OBJ    = 0
RTRLEN       = 0x0B   ; Write configuration or
DATA   = mode byte
DATA+1       = 0
```

```
x x x x  x x x x
      |  --- ---
      |   |   |------- 00 = do nothing
      |   |           01 = flush XHOLD buffer
      |   |           10 = disable ACK requirement
      |   |           11 = enable ACK requirement
      |   |
      |   |----------- 0x = do nothing
      |               10 = discard oldest packet on overflow
      |               11 = discard newest packet on overflow
      |
      |--------------- 1  = Master Reset (no acknowledge)
```

When read, the AccessMode function returns 2 bytes. The first byte is the ModeControl status, shown below. The second byte is the number of bytes in the HOLD buffer.

```
OBJ    = 0
RTRLEN       = 0x1B   ; Read configuration
DATA   = Mode Control
DATA+1       = Number of bytes in Hold Buffer
```

```
x x x x  x x x x
             | | |
             | | |------ if set, HOLD buffer waiting for an ACK
             | |
             | |-------- if set, HOLD buffer waits for ACK/NAK
             |
             |---------- if set, discard new message if XHOLD is
                         full
```

Code 12 -14 Reserved

Code 15 AccessStatus() allows the RS232 channel to access the last STATUS information or clear the XHOLD function to send the next packet.

The DIP065 sends two types of unsolicited messages: status and DATA (from CAN). When a WRITE_STATUS is received with the parameter byte == 0 it indicates that the last DATA packet has been acknowledged and the XHOLD function is free to send the next packet.

If the parameter byte == 0x1 it indicates that the last data byte was received incorrectly and should be retransmitted.

NOTE: ONLY DATA packets are expected to be acknowledged by the HOST.

NOTE: The WRITE_STATUS does not generate an explicit response. If the parameter byte is 0 then the implicit response occurs when the next CAN packet is received and transfered. If the parameter byte is 1 the implicit response occurs immediately by the retransmission of the previous packet. If the parameter byte > 1 then the function returns an E_PARAMETER message.

The StatusByte may be read. It has the following bit interpretations:

```
 x x x x  x x x x
      | | |
      | | |------ if set, HOLD buffer waiting for an ACK
      | |
      | |-------- if set, HOLD buffer waits for ACK/NAK
      |
      |---------- if set, discard new message if XHOLD is full
```

```
       OBJ         = 0
       RTRLEN      = 0x0F   ; Write configuration or
       DATA        = StatusByte

       OBJ         = 0
       RTRLEN      = 0x1F   ; Read configuration
       DATA        = Status Byte
```

Error Management

All packets sent by either the HOST or the DIP065 must be acknowledged. Configuration READ requests are acknowledged with either the requested data or an error packet. All other transactions are specifically acknowledged by a an error packet or an ACK packet.

To a large degree the error management is controlled by the HOST.

Synchronization

The HOST and DIP065 must maintain synchronization since the data bytes are transmitted in binary format. The units will re-synchronize upon receipt of 10 ETX characters. Note that this character burst may cause at least 1 NACK transaction to occur within the DIP065 module.

Chapter 4

Protocol Formats

Chapter Overview

This chapter will address the DIP065 Protocol Formats.

The transfer formats for data, configuration and acknowledgment packets share a common format consisting of a start of packet flag byte, the packet information, a checksum and an end of packet flag byte.

STX (Start of Packet)

All packets sent between the HOST and the DIP065 will start with an STX character. The STX character is the standard ASCII character 02H. The DIP065 module will ignore all RS232 data until the receipt of the STX character and will continue to buffer data until the receipt of the ETX character, up to a maximum of 11 characters (2 header + 8 data + checksum).

ETX (End of Packet)

All packets sent between the HOST and the DIP065 will end with an ETX character. The ETX character is the standard ASCII character 03H. The ETX character may be used to force synchronization.

CHECKSUM

All packets will include a 2's complement checksum of all packet information except the STX and ETX characters. The modulus 256 sum of all data + the checksum will be 0.

DATA AND CONFIGURATION PACKETS

DATA and CONFIGURATION packets are identical in format:

STX	OBJ	RTRLEN	<...DATA...>	CHKSUM	ETX

STX 02H

OBJ Object identifier bits 11-3 for data packets. Reserved field (0) for configuration packets.

RTRLEN Object identifier bits 2-0, RTR, Length (0-8) for data packets (see CAN Specifications). Command type code (9-15) for configuration packets.

DATA Variable length data. For data packets the length is encoded in the RTRLEN field. For configuration information the length is implicit in the configuration command byte.

CHKSUM Two's complement of OBJ, RTRLEN, DATA.

ETX 03H

ACKNOWLEDGEMENT PACKETS

Acknowledgement packets are used to verify transmission of previous packets and report failures.

STX	STATUS	CHKSUM	ETX

STX	02H
STATUS	Error code (see Errors).
CHKSUM	Two's complement of STATUS.
ETX	03H

ERROR CODES

The following error codes are defined.

ERROR	CODE	DESCRIPTION
E_OK	0	No Error. The last transmitted packet was accepted an fully processed. This is the ACK packet.
E_FULL	1	The RS232 to CAN receive buffer is in use due to a previous attempt to transmit. This error is generated upon receiving a transmit request before a previous transmit packet has been transferred to the CAN transmit buffer.
E_PARAM	2	An invalid configuration parameter was received in a command packet. The packet is ignored.
E_CHKSUM	3	A received checksum was incorrect.

DRV052 Functions

Chapter Overview

This chapter addresses the DRV052 functions (Windows).

For information on	See Page
DN Functions	5-1
CAN Functions	5-8
VXD Functions	5-10
Utility Functions	5-12
Error Codes	5-12
Visual Basic Function Prototypes	5-13

Note: drv052.dll was recompiled under the following names:
Drv052_c.dll (C calling convention) (C,C++)
Drv052_p.dll (Pascal calling convention) (Vbasic)

DN Functions

These functions allow the user to send DeviceNet commands:

- DNAllocate
- DNFree
- DNReset
- DNGetAttribute
- DNSetAttribute

DNAllocate

This function allows the user to create a M/S connection with a node within the DeviceNet network.

Function Prototype:

```
long DNAllocate (unsigned short int node, unsigned short int conn,
        unsigned char *buf)
```

Parameters:

node DeviceNet node that the user wants to allocate. The value ranges from 0 to 63.

conn Connection to be established with the node. (Explicit =1, Poll= 2, Strobe= 4, etc).

buf Pointer to an array of bytes for a response from DNAllocate. The size of the array must be 150.

C Declaration:

```
long rts;
int node;
int conn;
unsigned char buf[150];

rts = DNAllocate(node,conn,&buf);
```

Visual Basic Declaration:

```
Dim rts As Long                    ' return value
Dim node As Integer
Dim conn As Integer
Dim buf(150) as Byte

rts =DNAllocate(node,conn,buf(0))
```

Return Data:

*buf returns the following data

buf[0],[1]	Error code (0 for successful response)

Ignore the rest of the packet if an Error code is received.

buf[2],[3]	Receive Id from node
buf[4],[5]	Size of CAN message
buf[6],....	Message from node

Comments:

This function returns a non-zero value for Error. See Error Codes for details. The function waits 100ms for a response. If callback is implemented then the response is made available to the user as soon as it is received.

DNFree

This function allows the user to free M/S connection with a node within the DeviceNet network.

Function Prototype:

long DNFree (unsigned short int node, unsigned short int conn, unsigned char *buf)

Parameters:

node DeviceNet node that the user wants to free. The value ranges from 0 to 63.

conn Connection to be established with the node. (Explicit =1, Poll= 2, Strobe= 4).

buf Pointer to an array of bytes for function DNFree. The size of the array must be 150.

C Declaration:

```
long rts;
int node;
int conn;
unsigned char buf[150];

rts = DNFree(node,conn,&buf);
```

Visual Basic Declaration:

```
Dim rts As Long                         ' return value
Dim node As Integer
Dim conn As Integer
Dim buf(150) as byte

rts =DNFree(node,conn,buf(0))
```

Return Data:

*buf returns the following data

buf[0],[1]	Error code (0 for successful response)

Ignore the rest of the packet if an Error code is received.

buf[2],[3]	Receive Id from node

buf[4],[5]	Size of CAN message
buf[6],....	Message from node

Comments:

The function returns a non-zero value for Error. See Error Codes for details. The function waits 100ms for a response. If callback is implemented then the response is made available to the user as soon it is received.

DNReset

This function allows the user to reset the node.

Function Prototype:

long DNReset (unsigned short int node, unsigned short int clss, unsigned short int inst, unsigned short int rlen, unsigned char *buf)

Parameters:

node DeviceNet node that the user wants to reset. The value ranges from 0 to 63.

clss Class to be accessed.

inst Instance to be accessed.

rlen Number of characters send in *buf. Set to 0 if no data is to be sent.

buf On entry, buf has data to be sent to the node. On exit, buf has data response from DNReset. The size of the array must be 150.

C Declaration:

```
long rts;
int clss;
int inst;
int rlen;
unsigned char buf[150];

rts = DNReset(node,clss,inst,rlen,&buf);
```

Visual Basic Declaration:

```
Dim rts As Long                         ' return value
Dim clss As Integer
Dim inst as Integer
Dim rlen as Integer
Dim buf(150) as byte

rts =DNReset(node,clss,inst,rlen,buf(0))
```

Return Data:
*buf returns the following data

buf[0],[1]	Error code (0 for successful response)

Ignore the rest of the packet if an Error code is received.

buf[2],[3]	Receive Id from node
buf[4],[5]	Size of message
buf[6],....	Message from node

Comments:

The function returns a non-zero value for Error. See Error Codes for details. The function waits 100ms for a response. If callback is implemented then the response is made available to the user as soon it is received.

DNGetAttribute

This function supports DeviceNet Service GET_SINGLE.

Function Prototype:

long DNGetAttribute (unsigned short int node, unsigned short int clss, unsigned short int inst, unsigned short int attr, unsigned char *buf)

Parameters:

node DeviceNet node that the user wants to reset. The value ranges from 0 to 63.

clss Class to be accessed.

inst Instance to be accessed.

attr	Attribute to be accessed.
buf	On exit, buf has data response from DNGetAttribute. The size of the array must be 150.

C Declaration:

```
long rts;
int clss;
int inst;
int attr;
unsigned char buf[150];

rts = DNGetAttribute(node,clss,inst,attr,&buf);
```

Visual Basic Declaration:

```
Dim rts As Long                    ' return value
Dim clss As Integer
Dim inst as Integer
Dim attr as Integer
Dim buf(150) as byte

rts =DNGetAttribute(node,clss,inst,attr,buf(0))
```

Return Data:

*buf returns the following data

buf[0],[1]	Error code (0 for successful response)

Ignore the rest of the packet if an Error code is received.

buf[2],[3]	Receive Id from node
buf[4],[5]	Size of message
buf[6],....	Message from node

Comments:

The function returns a non-zero value for Error. See Error Codes for details. The function waits 100ms for a response. If callback is implemented then the response is made available to the user as soon it is received.

DNSetAttribute

This function supports DeviceNet Service SET_SINGLE.

Function Prototype:

```
long  DNSetAttribute(  unsigned short int node, unsigned short int
clss, unsigned short int inst, unsigned short int attr, unsigned short
int rlen, unsigned char *buf)
```

Parameters:

node DeviceNet node that the user wants to reset. The value ranges from 0 to 63.

clss Class to be accessed.

inst Instance to be accessed.

attr Attribute to be accessed.

rlen Number of characters send in *buf. Set to 0 if no data is to be sent.

buf On entry, buf has the data to be sent to the node. On exit, buf has data response from DNSetAttribute. The size of the array must be 150.

C Declaration:

```
long rts;
int clss;
int inst;
int attr;
int rlen;
unsigned char buf[150];

rts = DNSetAttribute(node,clss,inst,attr,&buf);
```

Visual Basic Declaration:

```
Dim rts As Long                 ' return value
Dim clss As Integer
Dim inst as Integer
Dim attr as Integer
Dim int as Integer
Dim buf(150) as byte
```

rts =DNSetAttribute(node,clss,inst,attr,rlen,buf(0))

Return Data:

*buf returns the following data

buf[0],[1]	Error code (0 for successful response)

Ignore the rest of the packet if an Error code is received.

buf[2],[3]	Receive Id from node
buf[4],[5]	Size of message
buf[6],....	Message from node

Comments:

The function returns a non-zero value for Error. See Error Codes for details. The function waits 100ms for a response. If callback is implemented then the response is made available to the user as soon it is received.

CAN functions

These functions allow the user to receive and transmit generic CAN messages.

CANRcv

This function will read a message from the VXD. If no messages are available an Error code is generated.

Function Prototype:

long CANRcv (unsigned short int *radd, unsigned short int *rlen, unsigned char *buf)

Parameters:

radd 11-bit identifier.

rlen Number of characters received.

buf On exit, data response from CANRcv. The size of the array must be 150.

C Declaration:

long rts;
int radd;

```
int rlen;
unsigned char buf[150];

rts = CANRcv(radd,rlen,&buf);
```

Visual Basic Declaration:

```
Dim rts As Long                  ' return value
Dim radd As Integer
Dim rlen as Integer
Dim buf(150) as byte

rts =CANRcv(radd,rlen,buf(0))
```

Return Data:

*buf returns the following data

buf[0],[1]	Error code (0 for successful response)

Ignore the rest of the packet if an Error code is received.

buf[2],[3]	Receive Id from node
buf[4],[5]	Size of message
buf[6],....	Message from node

Comment:

The function returns a non-zero value for Error. See Error Codes for details.

CANXmit

This function will write a message to the VXD.

Function Prototype:

```
long  CANXmit (unsigned short int xadd, unsigned short int xlen, unsigned char *buf);
```

Parameters:

radd 11-bit identifier.

xlen Number of characters received. This value must be less or equal to 8.

buf On entry, data to be sent to the node.

C Declaration:

```
long rts;
int xadd;
int xlen;
unsigned char buf[150];

rts = CANXmit(xadd,xlen,&buf);
```

Visual Basic Declaration:

```
Dim rts As Long                        ' return value
Dim xadd As Integer
Dim xlen as Integer
Dim buf(150) as byte

rts =CANXmit(xadd,xlen,buf(0))
```

Return Value:
NONE

VXD functions

LoadVXD

This function loads the proper driver for the DIP052 and DIP065. The DRV052.DLL can be used on Windows 95 and NT operating system.

Function Prototype:

```
long  LoadVXD  (unsigned short int Port, unsigned short int IRQ,
                unsigned char *Config);
```

Configuring DIP065:

COMM	ADDRESS
	Port
COM 1	2F8H
COM 2	3F8H

The 4-byte configuration array consists of the following UNSIGNED CHAR fields:

Config[4] accept_code.

accept_code Message IDENTIFIER(s) to be recognized by this node. Defines which message packets received by the controller will be accepted, subject to mask_code operation.

Config[5] mask_code.

mask_code MASK value which will be applied to accept_code and Message IDENTIFIER when qualifying message acceptance. The mask_code value is 'AND'ed with both the incoming message IDENTIFIER and the accept_code. Setting a bit within the mask_code informs the controller to ignore the corresponding bit in the accept_code. A mask_code of 0xFF will allow the controller to receive all packets.

Config[6] Bus Time 0

Bus Time 0 Baud rate multiplier and jitter correction control bits. (Refer to x32 specific register information).

Config[7] Bus Time 1

Bus Time 1 Data bit sampling control. (Refer to x32 specific register information).

To set up the data rate to 125kb use:

Bus Time 0	DEF_SPD125_0	0x03
Bus Time 1	DEF_SPD125_1	0x1c

To set up the data rate to 250kb use:

Bus Time 0	DEF_SPD250_0	0x01
Bus Time 1	DEF_SPD250_1	0x1c

To set up the data rate to 500kb use:

Bus Time 0	DEF_SPD500_0	0x00
Bus Time 1	DEF_SPD500_1	0x1c

To set up the data rate to 1000kb use:

Bus Time 0	DEF_SPD1000_0	0x00
Bus Time 1	DEF_SPD1000_1	0x14

Port	0x2f8, 0x3f8 for DIP065
IRQ	IGNORED
Config	As described above

UnloadVXD

This function unloads the proper driver for the DIP052 and DIP065. The DRV052.DLL can be used on Windows 95 and NT operating system.

Function Prototype:

```
long UnloadVXD();
```

Utility functions

These functions allow the user to make some simple conversion between data types.

The first set of functions converts bytes into integers, longs or floats by pointing to an element of the array.

```
unsigned short int Byte2Int (unsigned char *bData);
long  Byte2Long  (unsigned char *bData);
float Byte2Float (unsigned char *bData);
```

The second set of functions converts integers, longs or floats into bytes. These functions return 0.

```
long  Int2Byte   (unsigned short int *Param1, unsigned char
*bData);
long  Long2Byte  (unsigned long *Param1, unsigned char *bData);
long  Float2Byte (float *Param1, unsigned char *bData);
```

Error Codes

All user interface functions will return status information in the form of an unsigned long. The following are possible error codes.

E_OK	0x00	- No error detected.
E_TIMEOUT	0xffff	- Timed out due to lack of response.
E_NOTCONFIG	0xfffe	- DIP052 has not been configured.
E_BUSY	0xfffd	- DIP052 controller not available.
E_EMPTY	0xfffc	- No messages in receive queue.
E_FULL	0xfffb	- Transmit queue is full.
E_PRESENT	0xfffa	- DIP052 not present at specified port.
E_LENGTH	0xfff9	- length parameter incorrect.
E_PRESENT	0xfff8	- Unable to determine OS.
E_LENGTH	0xfff7	- Generic error.
E_LENGTH	0xfff5	- COM port is in used by another device.

Visual Basic Function Prototypes

The following section describes the declaration under Visual Basic.

Function Prototypes and Declaration:

Declare Function DNAllocate Lib "drv052.dll" (ByVal node As Integer, ByVal conn As Integer, rbuf As Any) As Long

Declare Function DNFree Lib "drv052.dll" (ByVal node As Integer, ByVal conn As Integer, rbuf As Any) As Long

Declare Function DNReset Lib "drv052.dll" (ByVal node As Integer, ByVal cls As Integer, ByVal inst As Integer, ByVal rlen As Integer, rbuf As Any) As Long

Declare Function DNGetAttribute Lib "drv052.dll" (ByVal node As Integer, ByVal cls As Integer, ByVal inst As Integer, ByVal attr As Integer, rbuf As Any) As Long

Declare Function DNSetAttribute Lib "drv052.dll" (ByVal node As Integer, ByVal cls As Integer, ByVal inst As Integer, ByVal attr As Integer, ByVal rlen As Integer, rbuf As Any) As Long

Declare Function CANRcv Lib "drv052.dll" (radd As Integer, rlen As Integer, rbuf As Any) As Long

Declare Function CANXmit Lib "drv052.dll" (ByVal radd As Integer, ByVal rlen As Integer, rbuf As Any) As Long

Declare Function LoadVXD Lib "drv052.dll" (ByVal port As Integer, ByVal Irq As Integer, config As Any) As Long

Declare Function UnloadVXD Lib "drv052.dll" () As Long

Declare Function Byte2Int Lib "drv052.dll" (xbuf As Any) As Integer

Declare Function Byte2Long Lib "drv052.dll" (xbuf As Any) As Long

Declare Function Byte2Float Lib "drv052.dll" (xbuf As Any) As Single

Declare Function Int2Byte Lib "drv052.dll" (par1 As Integer, xbuf As Any) As Long

Declare Function Long2Byte Lib "drv052.dll" (par1 As Long, xbuf As Any) As Long

Declare Function Float2Byte Lib "drv052.dll" (par1 As Single, xbuf As Any) As Long

Obtaining Help

Chapter Overview

This chapter will focus on obtaining help with the product.

For information on	See Page
Sources for Help	6-1

Sources for Help

Sources for obtaining help are listed below.

- ⌦ Visit the DIP Web Site at **http://www.dipinc.com**. The newest updates and revisions to the software as well as the documentation will be posted there.

- ⌦ Send a request for information through e-mail to **info@dipinc.com**. If the question is related to sales or marketing, send your e-mail to **sales@dipinc.com**.

- ⌦ Reach us by telephone at **(909) 686-4211**.

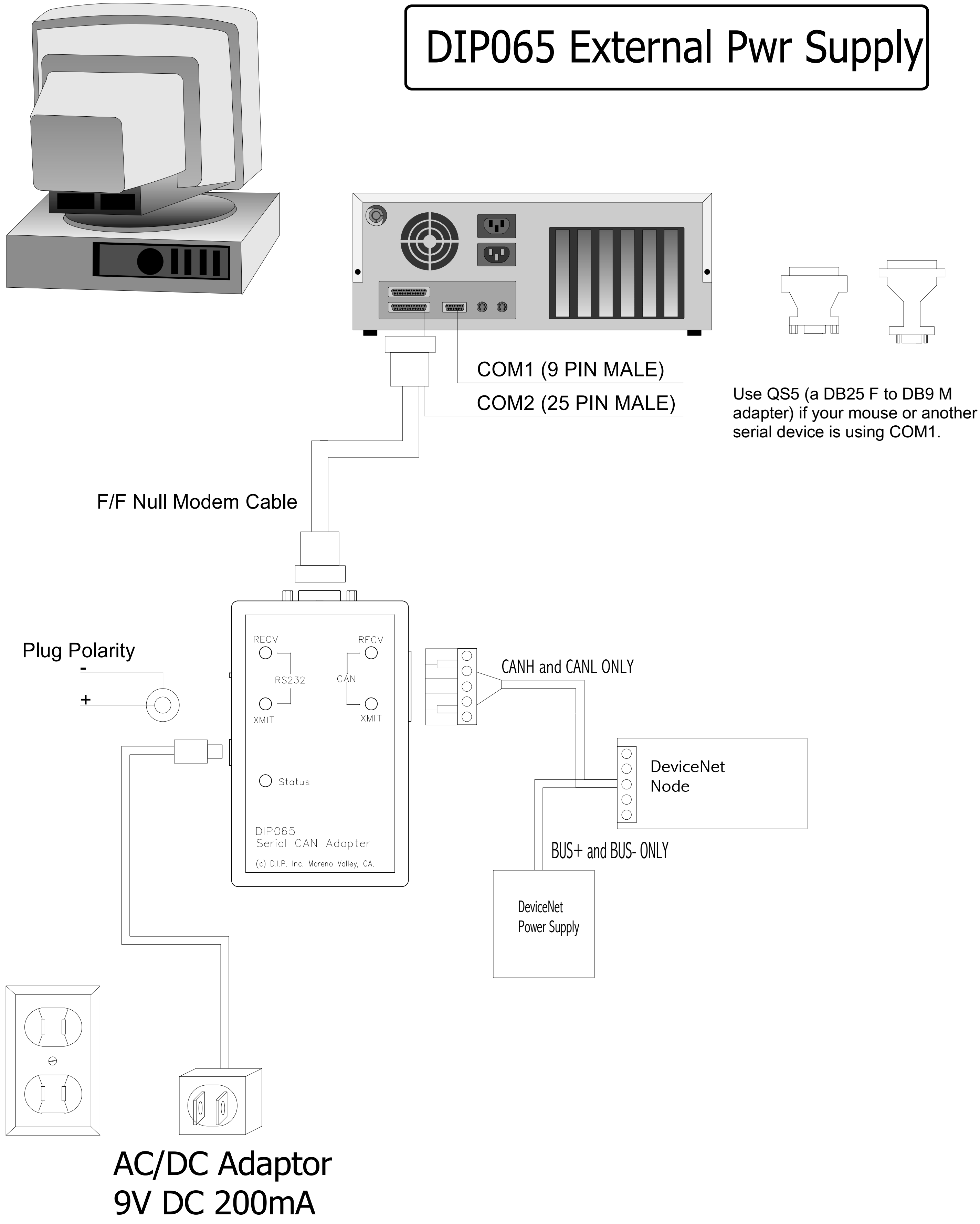
- ⌦ Fax us at **(909) 686-4122**.

- ⌦ Send us Postal Mail at:

DIP, Inc.
1860 Chicago Ave. Suite I-5
Riverside, CA 92507
USA

Hardware Installation Instructions

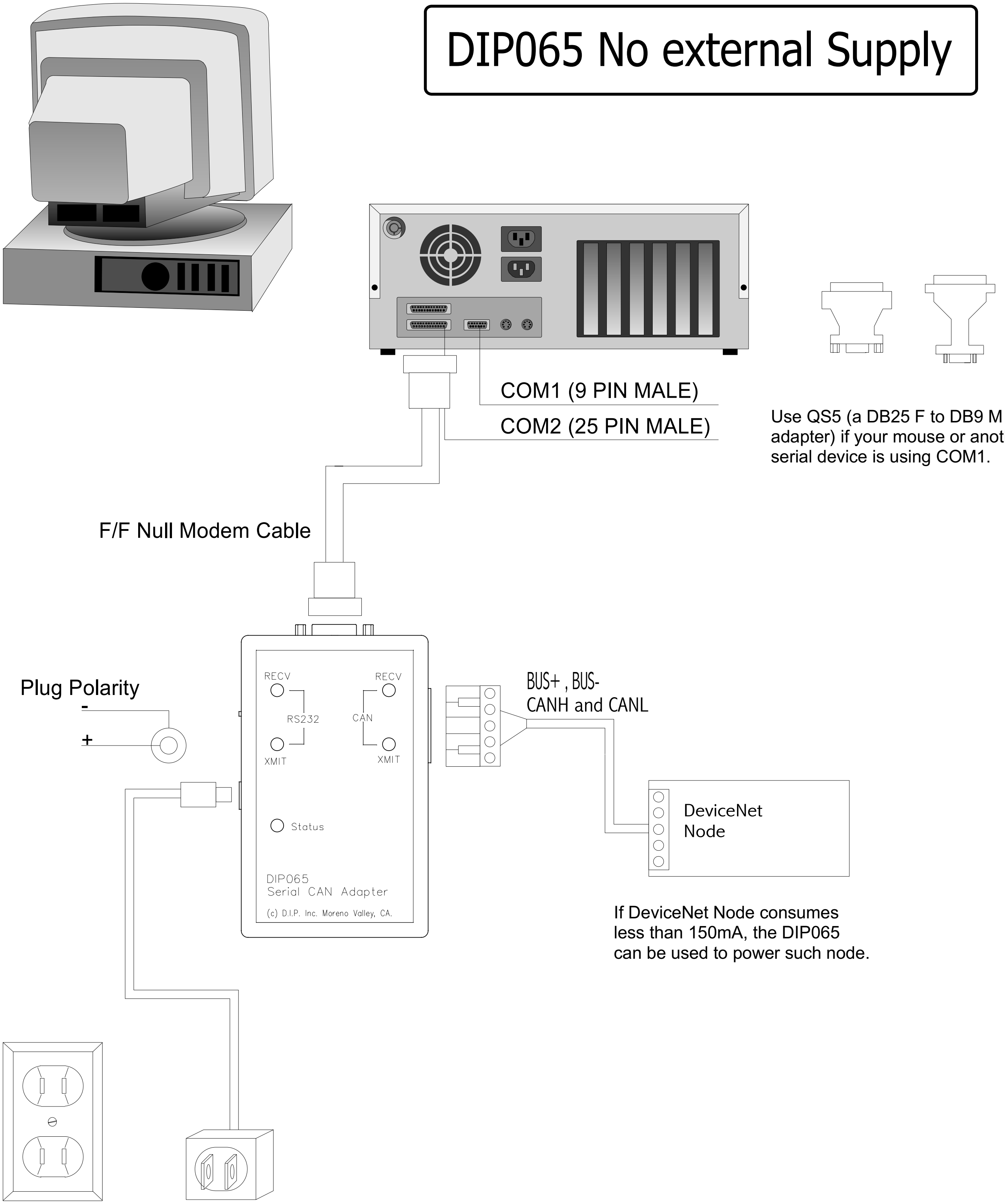
DIP065 External Pwr Supply



Use QS5 (a DB25 F to DB9 M adapter) if your mouse or another serial device is using COM1.

Hardware Installation Instructions

DIP065 No external Supply



Use QS5 (a DB25 F to DB9 M adapter) if your mouse or another serial device is using COM1.

If DeviceNet Node consumes less than 150mA, the DIP065 can be used to power such node.

AC/DC Adaptor
9V DC 200mA