

# CAN- $\mu$ VCCM USER MANUAL

Document Ver 1.03  
Software Ver 1.02  
Hardware Ver 1.00



*Copyright © 1999-2003*

*Acacetus Inc.*

*[www.acacetus.com](http://www.acacetus.com)*



# **CONTENTS**

<b>1</b>	<b>INTRODUCTION .....</b>	<b>4</b>
<b>2</b>	<b>DEVICE CONFIGURATION.....</b>	<b>5</b>
2.1	CONFIGURATION OVERVIEW .....	5
2.2	TERMINAL INTERFACE SETUP.....	5
2.3	ENTERING CONFIGURATION MODE .....	6
2.3.1	<i>First-Time Entry</i> .....	6
2.3.2	<i>User Initiated Entry</i> .....	6
2.4	INPUT EDITING KEYS.....	6
2.5	SAVING CONFIGURATION SETTINGS.....	7
2.6	CONFIGURING THE COM PORT.....	7
2.7	CONFIGURING THE CAN PORT .....	7
2.7.1	<i>Defining The CAN Bit-Time</i> .....	8
2.7.1.1	Setting the 'SYNC' Parameter .....	9
2.7.1.2	Setting the 'T_Prop' Parameter .....	9
2.7.1.3	Setting the 'T_SEG_1' Parameter .....	9
2.7.1.4	Setting the 'T_SEG_2' Parameter .....	9
2.7.1.5	Setting the 'SJW' Parameter.....	9
2.7.1.6	Setting the 'Tq Scaler' Parameter.....	9
2.7.1.7	Parameter Constraints.....	10
2.7.2	<i>Slew-Rate Adjustment</i> .....	11
2.7.3	<i>CAN Bus State During Configuration</i> .....	11
2.8	CONFIGURING DEVICE COMMANDS .....	12
2.8.1	<i>CMD_SERNUM</i> .....	12
<b>3</b>	<b>COMMAND MODE OPERATION .....</b>	<b>13</b>
3.1	FUNCTIONAL OVERVIEW .....	13
3.2	MESSAGE STRING SYNTAX.....	14
3.2.1	<i>Normal CAN Message</i> .....	15
3.2.1.1	Examples .....	15
3.2.2	<i>RTR CAN Message</i> .....	16
3.2.2.1	Examples .....	16
3.3	APPENDING CR/LF TO RECEIVED COMMAND STRINGS .....	17
3.4	USING MESSAGE FILTERS.....	17
3.4.1	<i>How Filtering Works</i> .....	17
3.4.2	<i>Setting Up Message Filters</i> .....	18
<b>4</b>	<b>VIRTUAL CIRCUIT MODE OPERATION.....</b>	<b>19</b>
4.1	FUNCTIONAL OVERVIEW .....	19
4.2	THE 'XMIT' AND 'RCEV' IDENTIFIERS .....	20
4.2.1	<i>'XMIT' Identifier</i> .....	20
4.2.2	<i>'RCEV' Identifier</i> .....	20
4.3	SPECIFYING INPUT STREAM TIMEOUT.....	20
4.3.1	<i>Normal Timeout Transmit Mode</i> .....	21



4.3.2	Immediate Transmit Mode.....	21
4.4	SPECIFYING TRANSMIT FORCE CODES .....	22
<b>5</b>	<b>DEVICE COMMAND OPERATION .....</b>	<b>23</b>
5.1	CMD_SERNUM .....	23
5.1.1	Functionality.....	23
5.1.2	CAN Port SERNUM Interaction.....	23
5.1.3	COM Port SERNUM Interaction.....	24
5.1.3.1	Example #1 .....	24
5.1.3.2	Example #2 .....	24
5.1.4	CAVEATS .....	25
<b>6</b>	<b>DIAGNOSTICS.....</b>	<b>26</b>
6.1	GENERATING CAN SQUARE WAVE .....	26
6.2	CYCLING LED INDICATORS .....	26
<b>7</b>	<b>HARDWARE SPECIFICATIONS.....</b>	<b>27</b>
7.1	POWER SUPPLY .....	27
7.2	COM RS232 SERIAL PORT.....	27
7.3	CAN NETWORK PORT .....	27
7.4	LED INDICATORS .....	27
7.5	SLEW-RATE ADJUST .....	27
<b>8</b>	<b>DOCUMENT CHANGE HISTORY .....</b>	<b>28</b>
8.1	VER 1.00 .....	28
8.2	VER 1.01 .....	28
8.3	VER 1.02 .....	28
8.4	VER 1.03 .....	28



# 1 INTRODUCTION

The CAN micro virtual-circuit and command-module ( $\mu$ VCCM) enables the use of a RS232 COM port to send and receive CAN messages on an ISO11898 CAN network (Figure 1).

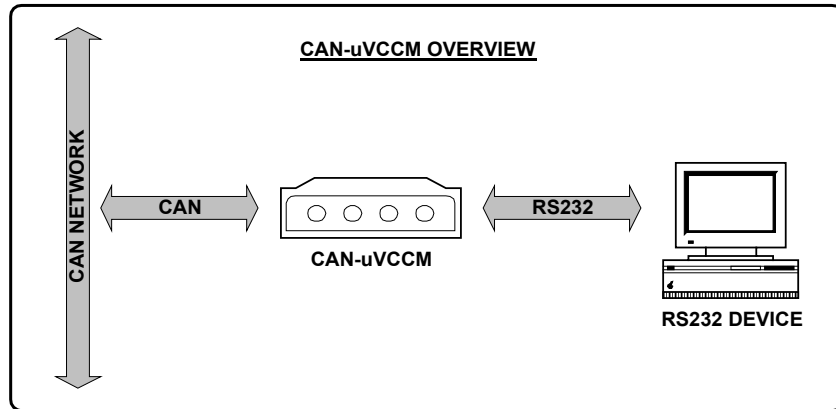


Figure 1

The  $\mu$ VCCM can be configured to operate in either the command mode or the virtual circuit mode, providing the user with a choice as to the type of functionality desired.

In the command mode, the  $\mu$ VCCM is capable of sending and receiving arbitrary CAN messages through the RS232 serial port, providing a complete RS232-CAN interface. This mode also supports message filtering to limit the range of CAN identifiers that will be received.

In the virtual circuit mode, the  $\mu$ VCCM creates a transparent 'virtual' RS232 link between itself and another  $\mu$ VCCM, providing the ability for applications to use a CAN network as a point-to-point RS232 link, supporting full-duplex exchange of arbitrary stream data.

In addition to the command and virtual-circuit modes presented above, the  $\mu$ VCCM also supports a user definable query for a 64-bit device serial number that is unique to each  $\mu$ VCCM unit. The query function is implemented as an RTR request on either the CAN or COM port via a user configured CAN identifier. Both the identifier to request the serial number and the identifier to respond with the information can be independently defined, allowing for flexible and efficient device enumeration protocols.

The  $\mu$ VCCM operating mode and all other configuration parameters are accessed through the RS232 serial port via a standard RS232 terminal. Upon power-up, the  $\mu$ VCCM looks for a specific input sequence on the RS232 COM port. If this sequence is detected in less than one second, the  $\mu$ VCCM enters configuration mode where it then prompts the user to modify configuration parameters. If the sequence is not detected before timeout, the  $\mu$ VCCM loads the saved settings from EEPROM and then begins normal execution.



In addition to parameter configuration, the configuration mode also provides diagnostic tools to assist in the measurement of bus signal propagation time, termination characteristics, and slew-rate adjustment.

## 2 DEVICE CONFIGURATION

### 2.1 Configuration Overview

In order to make the configuration process easy and efficient, the configuration program partitions the process into four sections.

The first section provides for changing the RS2323 COM port parameters such as baud, operating mode (VC or CM), etc.

The second section provides for changing the CAN bit definition parameters such as propagation time, phase segments 1 & 2, and set-jump-width.

The third section provides command configuration. Currently, only the device serial number command is supported. This section provides for the configuration of the serial number query command, allowing the user to enable/disable the feature and define the request and respond identifiers.

The fourth section provides for executing diagnostics functions such as generating a square wave on the CAN bus to assist in performing bus measurements, etc.

Each section is prompted with a yes/no request to the user and allows the user to skip configuration of a section if nothing is to be changed. This eliminates the need for the user to pass through all options when only a few must be changed.

### 2.2 Terminal Interface Setup

In order to configure the  $\mu$ VCCM, the user must connect a RS232, 3-wire, DTE terminal device to the  $\mu$ VCCM COM port.

The terminal must be configured for 9600 baud, 8-bits, no-parity, and 1 stop bit.

The terminal must also support the ASCII **<BS>**, **<CR>**, **<LF>**, **<ESC>**, and **<BELL>** control codes.

If possible, configure the terminal not to automatically generate a **<LF>** for each **<CR>** received, as the  $\mu$ VCCM will explicitly issue a **<CR>** **<LF>** sequence for each end-of-line it generates.



## 2.3 Entering Configuration Mode

### 2.3.1 First-Time Entry

The  $\mu$ VCCM is shipped from the factory with the EEPROM settings erased. When power is first applied to the  $\mu$ VCCM, default settings are stored in the EEPROM and configuration mode is automatically entered.

This ensures that the saved configuration is valid and that the user is aware of the configuration settings.

All subsequent configuration entries will operate as explained in section 2.3.2.

### 2.3.2 User Initiated Entry

Upon power-up, the  $\mu$ VCCM scans for a sequence of three ASCII **<SPACE>** characters (\$20) that must be received before a one-second timeout expires. If the sequence is detected before the timeout, the  $\mu$ VCCM enters configuration mode. If the sequence is not detected before the timeout, then configuration settings are loaded from EEPROM and the  $\mu$ VCCM begins normal operation. If any other input sequence is detected, the  $\mu$ VCCM will automatically load EEPROM settings and enter the normal operating mode.

Upon entering configuration mode, the  $\mu$ VCCM will display a banner indicating the product description, firmware revision, and copyright notice. It will then prompt the user to begin setting configuration parameters.

## 2.4 Input Editing Keys

To facilitate the editing of configuration parameters, certain keys are assigned some basic editing functions. Depending on the type of input variable being edited, some of these keys are redefined to make entry simpler.

The following editing keys are common to all configuration parameter types:

- <ESC>**       => Abort changes and restart from the beginning.
- <TAB>**        => Restore current parameter to state before any editing occurred.
- <SPACE>**     => Erase current input and move cursor to beginning of input line.
- <ENTER>**     => Update the current parameter and proceed to the next one.

For parameters that are either a Boolean variable or a list of choices, the following change in operation is in order:

- <SPACE>**     => Toggle the Boolean variable or advance to next item in list.

For all non-configuration parameters, the acceptable input keys are displayed at the time of the input prompt.



## 2.5 Saving Configuration Settings

Once the CAN and COM settings have been processed, the  $\mu$ VCCM prompts the user to either save the settings to EEPROM or re-start.

Choosing to save the settings will cause the  $\mu$ VCCM to write the new settings to EEPROM and will then prompt the user with diagnostics options.

Choosing to abort the save will re-start the configuration process from the top, using the initial settings at the time of the first editing.

## 2.6 Configuring The COM Port

Configuration of the COM port is simple and consists of specifying the RS232 serial BAUD rate, the number of data bits, parity, stop bits, and the mode of operation (Virtual Circuit or Command Mode).

The BAUD rate can range from 4800 to 57600 bits/sec.

The number of data bits can range from 7 to 8.

Parity can be odd, even, mark, space, or none.

The number of stop bits can be set to 1 or 2.<sup>1</sup>

Depending on the mode of operation selected, different configuration parameters will be displayed for change. Details of these parameters, their ranges, and their meanings can be found in the sections further below which describe the operating modes in greater detail.

## 2.7 Configuring The CAN Port

The CAN configuration parameters consist of propagation delay, phase segment #1 and #2, and the set-jump-window (SJW) parameters as specified in the CAN 2.0A/B specification.

The  $\mu$ VCCM will present the user a visual display of the current CAN bit setting to make it easier for the user to see just how a bit is defined. It will then prompt the user to make changes to the CAN parameters. Once all of the parameters have been changed or accepted, the visual display is redrawn and the user is prompted to accept, abort, or reenter the new settings.

The  $\mu$ VCCM also displays the 'Tq' time for the current  $\mu$ VCCM hardware. This is the basic unit used to define the CAN bit. Depending on the scale setting, this time can be increased in integer multiples, allowing for slower bit-rates.

Additional information such as the rules for the inter-relationship of the CAN parameters, the information processing time, and the combined RX/TX propagation delay of the CAN transceiver chip are also displayed.

---

<sup>1</sup> Depending on the number of data bits and parity selected. Data + parity + stop bits  $\leq$  10.



## 2.7.1 Defining The CAN Bit-Time

In order for a CAN network to correctly arbitrate multiple senders and to adapt to variances in system clocks, the low-level bit must be correctly specified for the specific characteristics of the network. These characteristics include cable length, desired data rate, transceiver propagation delay, information processing time (IPT), and accuracy of the oscillators used in the connected nodes. The  $\mu$ VCCM provides a means of configuring these parameters so as to support a wide array of network scenarios.

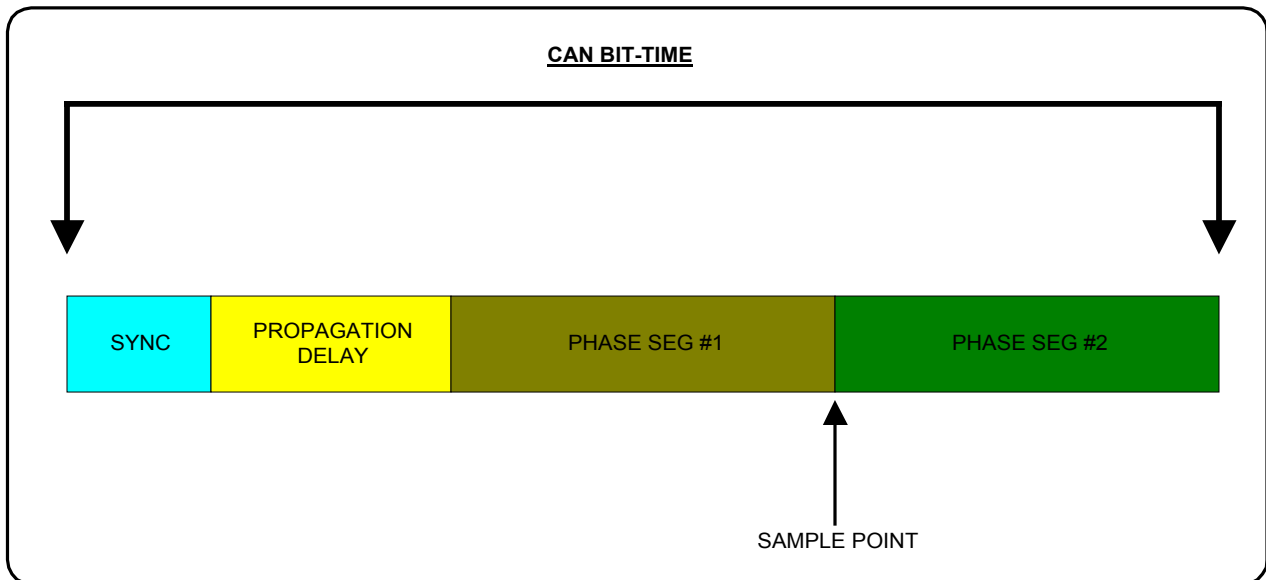


Figure 2

Figure 2 provides a visual example of the different parts of the CAN bit.

The CAN bit sections are defined in terms of integer 'time quanta' ( $T_q$ ). These  $T_q$  are fixed duration time intervals and are used as the basic time units when specifying the various parts of the bit time.

For the  $\mu$ VCCM hardware, the following times are specified:

- ❑ Minimum  $T_q$  (Scaler = 1)                      50 ns
- ❑ CAN Transceiver Delay (TX + RX)            350 ns
- ❑ CAN Information Processing Time            2  $T_q$

The following sections elaborate on the various bit time parameters:



### **2.7.1.1 Setting the 'SYNC' Parameter**

The 'SYNC' field is fixed in hardware and is always one Tq in duration.

### **2.7.1.2 Setting the 'T\_Prop' Parameter**

The 'T\_Prop' field represents the total, bi-directional propagation delay between the two furthest nodes on a network. It includes the TX & RX delays of the CAN transceivers, the signal propagation delay of the network cable, and the maximum information processing time of the CAN receiver module.

The value can range from 1 to 8.

### **2.7.1.3 Setting the 'T\_SEG\_1' Parameter**

The 'T\_SEG\_1' parameter determines the amount of time to wait before the hardware attempts to sample the bit.

The value can range from 1 to 8.

### **2.7.1.4 Setting the 'T\_SEG\_2' Parameter**

The 'T\_SEG\_2' parameter determines the amount of time remaining before the end of the bit.

The value can range from 1 to 8.

### **2.7.1.5 Setting the 'SJW' Parameter**

While not shown in Figure 2, the SJW parameter controls the maximum allowable adjustment to the sampling point. This allows for a CAN node to adjust its sampling point when it determines that the sample point would be too early or too late.

By setting the SJW field to a large number, the  $\mu$ VCCM can work with other nodes with a large variation in bus oscillator tolerances.

The value can range from 1 to 4.

### **2.7.1.6 Setting the 'Tq Scaler' Parameter**

As described above, the 'Tq Scaler' parameter determines the duration of the basic unit of time (Tq). When set to one, a Tq unit is 50 ns in duration.

The actual duration of a Tq is equal to the basic 50ns x 'Tq Scaler', providing the ability to define slower bit rates.

The value can range from 1 to 64.



### 2.7.1.7 Parameter Constraints

There are a few constraints on the possible combinations of the above parameters. The following rules must be respected to ensure correct CAN network operation:

- ❑ 'T\_SEG\_2' must be greater than or equal to two.
- ❑ 'T\_SEG\_2' must be greater then or equal to 'SJW'.
- ❑ 'T\_SEG\_2' must be less than or equal to 'T\_PROP' + 'T\_SEG\_1'.



## 2.7.2 Slew-Rate Adjustment

The  $\mu$ VCCM CAN transceiver supports the ability to adjust the slew-rate of the output CAN\_H and CAN\_L lines, reducing electrical noise at the expense of reducing the maximum bit-rate.

The slew-rate is adjusted by rotating the slew-adjust potentiometer until the desired slew-rate is in effect.

When the slew-rate potentiometer is fully counter-clockwise slew-rate control is disabled and the CAN transceiver will operate at its maximum rated speed of 1 mbit/sec.

A very simple method of adjusting the slew-rate is to first place the  $\mu$ VCCM into configuration mode and then select the 'generate square wave' option from the diagnostics menu. Choose the frequency closest to the one you will be working with and then connect the CAN\_H and CAN\_L lines to an oscilloscope and measure the slew rate. By varying the slew-adjust potentiometer, you can observe the changing slew-rate on the oscilloscope and adjust it accordingly until you have the rate you desire.

## 2.7.3 CAN Bus State During Configuration

While the  $\mu$ VCCM is in the configuration mode, the CAN controller is in the bus-off state. It does not interact with the bus and is effectively 'invisible'. Thus, it is possible to perform configuration of the  $\mu$ VCCM while still connected to an active CAN network and not adversely affect network operation.

Once the configuration state is exited, the  $\mu$ VCCM will activate the CAN controller and begin immediately interacting with the CAN network according to the CAN 2.0A/B specification.

It is the users responsibility to ensure that the configuration settings are appropriate for the network to which the  $\mu$ VCCM is connected so that the  $\mu$ VCCM operates correctly and does not cause erratic network operation.



## 2.8 Configuring Device Commands

The  $\mu$ VCCM is designed to support user commands concurrently with virtual-circuit and command mode operation. The following sections explain how to configure the device commands.

### 2.8.1 CMD\_SERNUM

The SERNUM command is a request/response transaction where the user issues a REQUEST CAN message and the  $\mu$ VCCM responds with a RESPOND CAN message containing its unique 64-bit number in the eight data bytes of the response message.

Through the configuration process, the user can enable or disable the SERNUM command.

If enabled, the user is prompted to specify which CAN identifier to use as the REQUEST message and which to use as the RESPOND message. Both the identifier values and sizes (11 or 29 bit) can be specified.

The identifiers can be identical since the REQUEST identifier will always be in a RTR message and the RESPOND identifier will be in a non-RTR message.

If VC mode is enabled, the configuration software will not allow you to specify REQUEST or RESPOND identifiers that conflict with the VC XMIT and RCEV identifiers in order to ensure that the unit does not corrupt CAN communication on the network by sending conflicting or misinterpreted messages.



### 3 COMMAND MODE OPERATION

#### 3.1 Functional Overview

In the command mode, the  $\mu$ VCCM is capable of sending and receiving arbitrary CAN messages via the use of ASCII formatted message strings on the RS232 port (Figure 3).

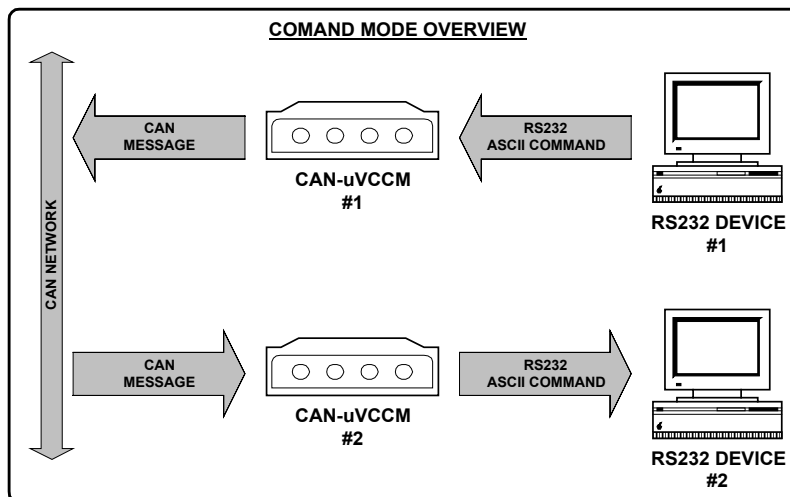


Figure 3

When the  $\mu$ VCCM receives a valid ASCII message string, it converts it to a CAN message and transmits it out over the CAN network.

Conversely, when a CAN message is received by the  $\mu$ VCCM, it converts it to an ASCII message string and transmits it out of the RS232 port.

The  $\mu$ VCCM supports CAN receive message filtering through the use of four identifier masks and filters. By using the masks to specify which bits of an identifier are to be compared to the filter value, the  $\mu$ VCCM is capable of selecting an arbitrary sub-set of the total possible CAN messages and rejecting all others. Thus, only desired messages will be received and the total required bandwidth of the RS232 link is kept to a minimum.

In order to facilitate human CAN network monitoring, there is an option to append a CR/LF sequence to each output ASCII message string. Doing so makes it much easier to watch the incoming messages on a terminal where each message is on a separate line.



### **3.2 Message String Syntax**

Message strings are formatted as human-readable ASCII sequences that are easy to enter and read. Each message string is of variable length, depending on the identifier value and the number of data bytes included in the message.

All message string characters are in upper case only. Lower case characters will be interpreted as a syntax error and the message will be discarded.

Identifier and data fields are treated as base-16 digits (hexadecimal).

The length field is treated as a base-10 digit and must be a single digit between 0 and 8.

The syntax of both transmit and receive message strings are identical.

There are two types of message strings:

- Normal CAN messages
- Request-To-Transmit CAN messages (RTR)

As per the CAN 2.0A/B specification, RTR messages do not contain data. To support RTR messages, the syntax is modified to include an explicit single-digit length.



### 3.2.1 Normal CAN Message

A normal CAN message consists of the type (11-bit or 29-bit), identifier, length, and data bytes and is encoded as follows:

#### Normal CAN Message Syntax

```
: <S | X> <IDENTIFIER> <N> <DATA-0> <DATA-1> ... <DATA-7> ;
```

The first character, ':', is for synchronization and allows the  $\mu$ VCCM parser to detect the beginning of a command string.

The following character is either 'S' for standard 11-bit, or 'X' for extended 29-bit identifier type.

The 'IDENTIFIER' field consists of from one to eight hexadecimal digits, indicating the value of the identifier. Note that if a 29-bit value is entered and an 11-bit value was specified, the command will be treated as invalid and ignored.

The character 'N' indicates that the message is a normal (non-RTR) transmission.

Each 'DATA-n' field is a pair of hexadecimal digits defining the data byte to be sent. If no data is to be sent (length = zero), then the data bytes are omitted.

Each data byte specified must be a hexadecimal pair in order to eliminate ambiguity.

The terminating character ';' signals the end of the message.

#### 3.2.1.1 Examples

Example #1

```
:S123N12345678;
```

This message string indicates a 11-bit identifier whose value is \$123, is normal, and has four data bytes : \$12 \$34 \$56 and \$78.

Example #2

```
:XF00DN;
```

This message string indicates a 29-bit identifier whose value is \$F00D, is normal, and has zero data bytes.



### 3.2.2 RTR CAN Message

A RTR CAN message consists of the type (11-bit or 29-bit), identifier, length, does not have any data bytes, and is encoded as follows:

#### RTR CAN Message Syntax

```
: <S | X> <IDENTIFIER> <R> <LENGTH> ;
```

The first character, ':', is for synchronization and allows the  $\mu$ VCCM parser to detect the beginning of a command string.

The following character is either 'S' for standard 11-bit, or 'X' for extended 29-bit identifier type.

The 'IDENTIFIER' field consists of from one to eight hexadecimal digits, indicating the value of the identifier. Note that if a 29-bit value is entered and an 11-bit value was specified, the command will be treated as invalid and ignored.

The character 'R' indicates that the message is a RTR transmission.

The 'LENGTH' field is a single ASCII decimal character from '0' to '8' that specifies the length of the message.

The terminating character ';' signals the end of the message.

#### 3.2.2.1 Examples

Example #1

```
:S123R8;
```

This message string indicates a 11-bit identifier whose value is \$123, is RTR, and is of length 8.

Example #2

```
:XF00DR0;
```

This message string indicates a 29-bit identifier whose value is \$F00D, is RTR, and is of length 0.



### 3.3 Appending CR/LF To Received Command Strings

When using the  $\mu$ VCCM to view received CAN messages directly on a terminal, the user can set a configuration parameter to append a <CR> <LF> sequence to each message string generated. This makes it easier for the user to see each received message as each message will be on a separate line.

### 3.4 Using Message Filters

In command mode, the  $\mu$ VCCM supports message filtering through the use of four identifier masks and filter values, providing the ability to receive only a sub-set of all the possible CAN identifiers. This can greatly reduce the required RS232 bandwidth on a busy network by only allowing certain messages to be received.

To use the filters, they must be enabled, their size must be specified, and appropriate mask & filter values must be assigned. This configuration is done while in the configuration mode as described in section 2.1 and while in the COM phase of configuration.

While there are four filters, it is possible that none, some, or all are being used.

#### 3.4.1 How Filtering Works

Whenever the  $\mu$ VCCM receives a CAN message from the network, it checks to see if any filters are enabled. If none of the filters are enabled, it assumes no filtering should be performed and outputs the ASCII message string of the message to the RS232 COM port. If any filters are enabled, the  $\mu$ VCCM will attempt to match the received message to each enabled filter. At the first successful match, the message is converted to an ASCII message string and output to the RS232 COM port. If no matches were successful, the message is discarded.

In order to determine if a received identifier matches a filter, the  $\mu$ VCCM first checks if the size of the identifier matches that of the filter. If either there is a direct match of the identifier size or the filter is configured to receive either size, then the  $\mu$ VCCM will proceed to the next test. If there is a size mismatch, the message is discarded.

Once the size test has passed, the  $\mu$ VCCM will attempt to match the identifier value to the filter value. It first performs a bit-wise 'AND' of the identifier and the filter mask, leaving only the identifier bits of interest. It then compares the result of the 'AND' operation with the filters value to determine if the two are the same. If they are identical, the filter is considered to have matched to the identifier and the message is convert to an ASCII message string and sent out to the RS232 COM port. If not, the message is discarded.

The above sequence of operations is performed on each received identifier for all enabled filters.



### 3.4.2 Setting Up Message Filters

To use message filtering, each filter used must be enabled and then configured as to the type of filtering that is desired. All of these parameters are configured during the configuration process as explained in section 2.1.

A filter is configured in the following order:

- 1) Enable the filter
- 2) Specify if standard only, extended only, or either type of identifier will be matched.
- 3) Set the mask value.
- 4) Set the filter value.

The  $\mu$ VCCM will sequence through the filter configuration items for each filter until either the user disables a filter or all of the filters have been configured.

If a filter is disabled, all the remaining filters are also disabled, ensuring that they do not adversely affect message processing.

Once all configuration parameters have been saved and the  $\mu$ VCCM enters the normal operating mode, the new filter settings will become active and will be applied to each received CAN message as describe above in section 3.4.1.



## 4 VIRTUAL CIRCUIT Mode Operation

### 4.1 Functional Overview

In virtual circuit mode, the  $\mu$ VCCM establishes a full-duplex, virtual, RS232 circuit between itself and another  $\mu$ VCCM or application device. By providing a virtual RS232 circuit over the CAN network, applications can exchange RS232 stream data in a network-transparent fashion, using existing CAN network cabling as an RS232 link (Figure 4).

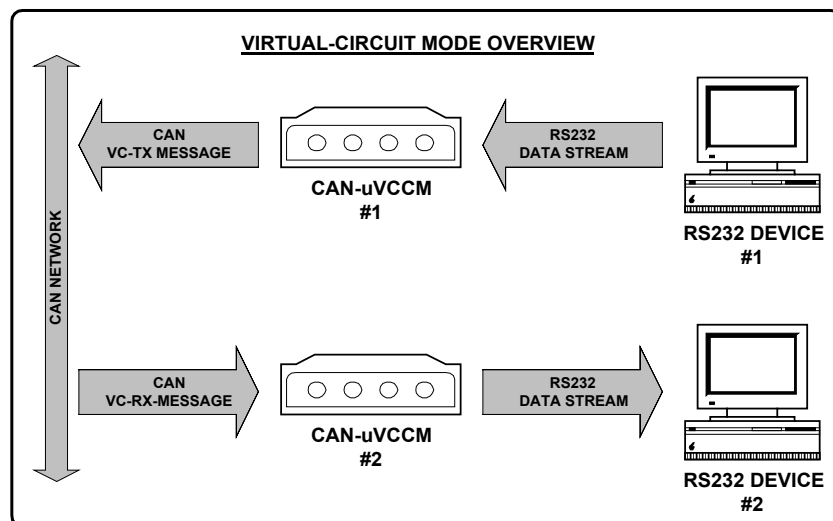


Figure 4

The  $\mu$ VCCM takes data bytes coming into the RS232 port and groups them into CAN messages for transmission. When a remote  $\mu$ VCCM or application target receives these messages, it extracts the data bytes and recreates the original data stream on its RS232 output port. This operation is fully transparent to the connected application devices.

The virtual circuit requires two, user-configurable, CAN identifiers in order to identify the source & destination devices for the circuit. Since virtually every CAN protocol provides a sub-set of unused or 'user-specific' identifiers, establishing a virtual circuit on a CAN network in conjunction with an existing protocol is supported and very easy to do.

The  $\mu$ VCCM only sends CAN messages when eight bytes of data have been received. This leads to a case where the last part of a data stream is not sent if it is less than eight bytes long. To deal with this in a transparent fashion, the  $\mu$ VCCM provides a timeout feature that will automatically force a transmission of the last accumulated byte(s) after a specified maximum waiting time.

The  $\mu$ VCCM can also be configured to force immediate transmission upon detection of specific user-configured bytes in the data stream (CR or LF, for example).



## 4.2 The 'XMIT' and 'RCEV' Identifiers

When configuring the  $\mu$ VCCM for Virtual Circuit operation, two identifiers must be specified in order for the virtual circuit to work correctly. These two identifiers are referred to as the 'XMIT' and 'RCEV' identifiers.

### 4.2.1 'XMIT' Identifier

The 'XMIT' identifier is used when the  $\mu$ VCCM has stream data to send over the CAN network. A CAN message is formatted to include this identifier, the number of bytes to be sent, and the actual stream data that was received from the RS232 COM port. The CAN message is then sent over the CAN network.

This identifier signals that the CAN message which contains it also contains serial port stream data. Other  $\mu$ VCCM units can be configured to look for this identifier and will then extract the stream data from the CAN message.

The 'XMIT' identifier can be either standard 11-bit or extended 29-bit.

### 4.2.2 'RCEV' Identifier

The 'RCEV' identifier is used to specify which CAN message should be received by the  $\mu$ VCCM when waiting for incoming stream data. The  $\mu$ VCCM will discard all other messages.

When a CAN message containing this identifier is received, the data in the message is assumed to be stream data from another  $\mu$ VCCM. This data is then read and output directly to the RS232 COM port of the receiving  $\mu$ VCCM unit, providing the completion of the virtual circuit.

If the received message for any reason contains a length of zero, or is an RTR message, it is discarded and not stream data is generated.

The 'RCEV' identifier can be either standard 11-bit or extended 29-bit.

## 4.3 Specifying Input Stream Timeout

As overviewed in section 4.1, the  $\mu$ VCCM supports a timeout feature when in virtual circuit mode and waiting for input data from the RS232 COM port. This timeout feature provides the ability to transparently handle the last few bytes of a serial data input stream, ensuring that these bytes are not waiting inside the  $\mu$ VCCM to be sent when no further input data is arriving.

This parameter is configured during in configuration mode under the 'COM' section as described in section 2.1.

The operating range for the timeout parameter is from zero 0 to 1000 ms.



#### **4.3.1 Normal Timeout Transmit Mode**

When the timeout range specified is from 1 to 1000 ms, the  $\mu$ VCCM will operate normally, waiting up to the maximum amount of time for new stream data on the RS232 COM port before transmitting a CAN message.

#### **4.3.2 Immediate Transmit Mode**

When the timeout value is set to zero (0), the  $\mu$ VCCM enters a special transmit mode where each byte received from the RS232 COM port is immediately transmitted as CAN message of length 1, using the 'XMIT' identifier as described in section 4.2.1.

This mode allows immediate response to input stream data at the expense of efficiency since each incoming stream byte now has the total overhead of a CAN message, whereas before, up to eight stream bytes could be packed into a single CAN message, increasing efficiency.



#### **4.4 Specifying Transmit Force Codes**

As overviewed in section 4.1, the  $\mu$ VCCM can detect up to two different data codes in the serial data input stream that will force immediate transmission of the currently accumulated stream bytes. When either of these codes is detected, the  $\mu$ VCCM will format a CAN message with the 'XMIT' identifier as explained in section 4.2.1 and immediately transmit it over the CAN network.

In order to use these codes, they must first be enabled and then specified.

To enable and specify the codes, enter the configuration mode under the 'COM' section and select the virtual circuit mode. The input prompts will prompt for enabling/disable force-transmit codes and, if enabled, will prompt for the ASCII codes to scan for.

Once configuration parameters are saved and the  $\mu$ VCCM enters normal operation, the new force codes will come into effect.



## 5 DEVICE COMMAND OPERATION

The  $\mu$ VCCM was designed to support the addition of commands that could be executed concurrently with either the virtual-circuit or command modes of operation. The following sections define which commands are currently implemented and how they operate.

Commands are defined as user-specified CAN messages that are enabled/disabled during configuration and perform various operations when the relevant CAN command message is received.

When a command is enabled, the  $\mu$ VCCM performs additional checking on CAN messages received on either the CAN or COM port and when it detects a defined and enabled command message, it processes it and issues a response (if applicable).

Currently, there is only one command supported: CMD\_SERNUM

### 5.1 CMD\_SERNUM

The SERNUM command provides a means for the user to query the  $\mu$ VCCM for its unique 64-bit serial number by issuing a user-defined CAN message to either the CAN or COM port, depending on the operating mode of the unit, the  $\mu$ VCCM will respond with a CAN message containing its 64-bit serial number.

#### 5.1.1 Functionality

When the user issues a CAN RTR message with the user-defined REQUEST identifier and message length set to eight bytes, the  $\mu$ VCCM will respond (if enabled) with a CAN message containing the RESPOND identifier and eight data bytes containing the 64-bit serial number.

For both the REQUEST and RESPOND identifiers, the size (11 or 29 bit) and value of the identifiers is fully configurable by the user.

In order to support multiple and simultaneous SERNUM queries, the REQUEST CAN message must always be sent as a RTR message with a length of eight bytes.

The RESPOND CAN message is never an RTR message and care must be taken to ensure that no two  $\mu$ VCCM units will be responding with the same identifier. Doing so would violate the rules of the CAN 2.0B specification and may lead to erratic and erroneous operation of the CAN network.

#### 5.1.2 CAN Port SERNUM Interaction

When a SERNUM command is detected on the CAN port and the SERNUM command is enabled, the  $\mu$ VCCM will always respond with a CAN message containing its serial number regardless of whether the COM port is in virtual-circuit or command mode.

The response is issued only to the CAN network and is not copied to the COM port.



### 5.1.3 COM Port SERNUM Interaction

When a SERNUM command is detected on the COM port and the SERNUM command is enabled, the  $\mu$ VCCM will respond differently, depending on which mode of operation it is in.

If the COM port is operating in the command mode, then the SERNUM query is processed and a response CAN message is transmitted to the COM port to the user.

If the COM port is operating in the virtual-circuit mode, then there is no way for the  $\mu$ VCCM to detect any commands since it is treating all input data as arbitrary data to be sent down the virtual circuit. However, the  $\mu$ VCCM will still respond to queries on the CAN port as described earlier.

The following examples show how COM port SERNUM queries work:

#### 5.1.3.1 Example #1

The SERNUM request identifier is defined to be \$123 and 11-bits in size, RTR.

The response identifier is also defined as \$123 and 11-bits in size.

The  $\mu$ VCCM is operating in CM mode and the user has sent it the following command:

```
:S123R8;
```

The  $\mu$ VCCM unit responds with the following message string:

```
:S123N1122334455667788;
```

This message string indicates a CAN message with identifier set to \$123, size is 11-bit, normal, and contains eight data bytes indicating the following 64-bit serial number:

```
$1122334455667788
```

#### 5.1.3.2 Example #2

The SERNUM request identifier is defined to be \$BEEF and 29-bits in size, RTR.

The response identifier is defined as \$303 and 11-bits in size.

The  $\mu$ VCCM is operating in CM mode and the user has sent it the following command:

```
:XBEEFR8;
```

The  $\mu$ VCCM unit responds with the following message string:

```
:S303N1122334455667788;
```

This message string indicates a CAN message with identifier set to \$303, size is 11-bit, normal, and contains eight data bytes indicating the following 64-bit serial number:

```
$1122334455667788
```

ACACETUS INC

[www.acacetus.com](http://www.acacetus.com)



#### **5.1.4 CAVEATS**

There are a couple of things to keep in mind when using the SERNUM command feature:

- In CM mode, SERNUM requests have priority over other CAN messages and will not be passed through. This means that a SERNUM message received on the CAN port will not be seen on the COM port and vice versa.
- In VC mode, SERNUM requests are only processed when received on the CAN port.
- In VC mode, the identifiers used for REQUEST and RESPOND must be different from those used by the virtual circuit XMIT and RCEV messages.
- Response message IDs must be unique for each  $\mu$ VCCM unit on the network.



## 6 Diagnostics

Once all of the user configurable options have been processed, the  $\mu$ VCCM prompts the user with the option to perform some diagnostics.

To bypass configuration, select 'N' when prompted to change settings. This will then bring up the diagnostics prompt, skipping the change procedure.

### 6.1 Generating CAN Square Wave

The  $\mu$ VCCM can generate a square wave output onto the CAN bus at various pre-defined frequencies. This square wave can be used to measure and adjust the slew rate via the slew rate adjust potentiometer. It is also useful for measuring propagation delay through cabling with the aid of an oscilloscope and can be used to evaluate bus termination.

The square wave frequency can be one of the following: 100, 125, 150, 200, 250, 500, or 1000 kbits/sec.

To access the generator, enter the configuration mode and select no changes. When the diagnostics prompt appears, select 'Y' for the 'Generate Square Wave' prompt.

To terminate generation, press any key.

### 6.2 Cycling LED Indicators

To confirm that all LED indicators are working, this diagnostic allows the user to cycle through the CAN-TX/RX and COM-TX/RX LED indicators one at a time.

As each indicator is illuminated, an ASCII message is transmitted out onto the COM port identifying the current LED being lit by name.

Each time the <SPACE> bar is pressed, the next LED is illuminated and its associated ASCII name is transmitted.

To access the LED diagnostic, enter the configuration mode and select no changes. When the diagnostics prompt appears, select 'Y' for the 'Cycle LED Indicators' prompt.

To terminate the diagnostic, press the <ENTER> key.



## 7 Hardware Specifications

### 7.1 Power Supply

- ❑ 7 to 60 Volts DC input from standard wall transformer.
- ❑  $\frac{3}{4}$  Watt : 150 ma

### 7.2 COM RS232 Serial Port

- ❑ RS232 3-Wire DCE
- ❑ Female DB9 Connector
- ❑ 4800 to 57600 Baud
- ❑ 7 or 8 data bits
- ❑ Odd, even, mark, space, or no parity
- ❑ 1 or 2 stop bits<sup>2</sup>
- ❑ Pinout is DCE (TX = 2 , RX = 3 , GND = 5).
- ❑ Receive message buffering : 2 Messages

### 7.3 CAN Network Port

- ❑ ISO11898 CAN Transceiver up to 1 mbit/sec.
- ❑ Electrical isolation up to 1000 VDC.
- ❑ Female DB9 Connector
- ❑ Pinout is C.I.A. DS-102 (CAN\_H = 7, CAN\_L = 2, CAN\_GND = 3).
- ❑ Receive message buffering : 10 Messages

### 7.4 LED Indicators

- ❑ CAN TX & RX activity.
- ❑ COM TX & RX activity.
- ❑ POWER on indicator.

### 7.5 Slew-Rate Adjust

- ❑ MAX  $\leq 20$  V/ $\mu$ s
- ❑ MIN  $\geq 2.5$  V/ $\mu$ s
- ❑ Adjustable via potentiometer and screwdriver.
- ❑ Built-in signal generator to aid in adjustment.

---

<sup>2</sup> Not all data bit length, parity, and stop bit combinations are valid. Data + parity + stop bits  $\leq 10$ .



## 8 DOCUMENT CHANGE HISTORY

### 8.1 Ver 1.00

- Initial release.

### 8.2 Ver 1.01

- Added firmware and hardware version reference numbers to document.
- Changed MIN input voltage from 8VDC to 7VDC.
- Changed MAX input voltage from 40VDC to 60VDC.

### 8.3 Ver 1.02

- Added documentation for new device serial number query feature (SERNUM).
- Minor spelling corrections and clarifications.

### 8.4 Ver 1.03

- Added documentation for new serial port configurations (7 or 8 data bits, parity, 1 or 2 stop bits).
- Minor corrections (4800 baud instead of 9600 baud).