gridconnect™

CAN WiFi
User Guide

Revision F.1 April 17, 2020

Copyright and Trademark

Grid Connect

1630 W. Diehl Rd.
Naperville, IL 60563, USA
Phone: 630.245.1445

Technical Support

Phone: 630.245.1445
Fax: 630.245.1717
On-line: www.gridconnect.com

Disclaimer and Revisions

Operation of this equipment in a residential area is likely to cause interference in which case the user, at his or her own expense, will be required to take whatever measures may be required to correct the interference.

> ***Attention:*** *This product has been designed to comply with the limits for a Class B digital device pursuant to Part 15 of FCC Rules. These limits are designed to provide reasonable protection against harmful interference in a residential installation. This equipment generates, uses, and can radiate radio frequency energy, and if not installed and used in accordance with this guide, may cause harmful interference to radio communications.*

Changes or modifications to this device not explicitly approved by Grid Connect will void the user's authority to operate this device.

The information in this guide may change without notice. The manufacturer assumes no responsibility for any errors that may appear in this guide.

Date	Rev.	Author	Comments
06/09/2014	A	GR	Preliminary Release
10/23/2014	B	GR	Removed references to CAN-USB232.
05/14/2015	C	GR	16-bit Time Stamp on received messages
03-02-2016	D	GR	General updates
08-18-2016	E	GR	Add new pictures and info for reset button
11-18-2016	F	GR	Skip count or timeout between consecutive CAN messages received.
04-17-2020	F.1	JK	Fixed voltage specs

Warranty

Grid Connect warrants each product to be free from defects in material and workmanship for a period of ONE YEAR after the date of shipment. During this period, if a customer is unable to resolve a product problem with Grid Connect Technical Support, a Return Material Authorization (RMA) will be issued. Following receipt of a RMA number, the customer shall return the product to Grid Connect, freight prepaid. Upon verification of warranty, Grid Connect will -- at its option -- repair or replace the product and return it to the customer freight prepaid. If the product is not under warranty, the customer may have Grid Connect repair the unit on a fee basis or return it. No services are handled at the customer's site under this warranty. This warranty is voided if the customer uses the product in an unauthorized or improper way, or in an environment for which it was not designed.

Grid Connect warrants the media containing software and technical information to be free from defects and warrants that the software will operate substantially for a period of 60 DAYS after the date of shipment.

In no event will Grid Connect be responsible to the user in contract, in tort (including negligence), strict liability or otherwise for any special, indirect, incidental or consequential damage or loss of equipment, plant or power system, cost of capital, loss of profits or revenues, cost of replacement power, additional expenses in the use of existing software, hardware, equipment or facilities, or claims against the user by its employees or customers resulting from the use of the information, recommendations, descriptions and safety notations supplied by Grid Connect. Grid Connect liability is limited (at its election) to:

1) refund of buyer's purchase price for such affected products (without interest)

2) repair or replacement of such products, provided that the buyer follows the above procedures.

There are no understandings, agreements, representations or warranties, expressed or implied, including warranties of merchantability or fitness for a particular purpose, other than those specifically set out above or by any existing contract between the parties. The contents of this document shall not become part of or modify any prior or existing agreement, commitment or relationship.

Table of Contents

1. Overview

1.1 Introduction

The CAN WIFI module is a hybrid device with two completely separate sub-systems connected internally via a UART interface.

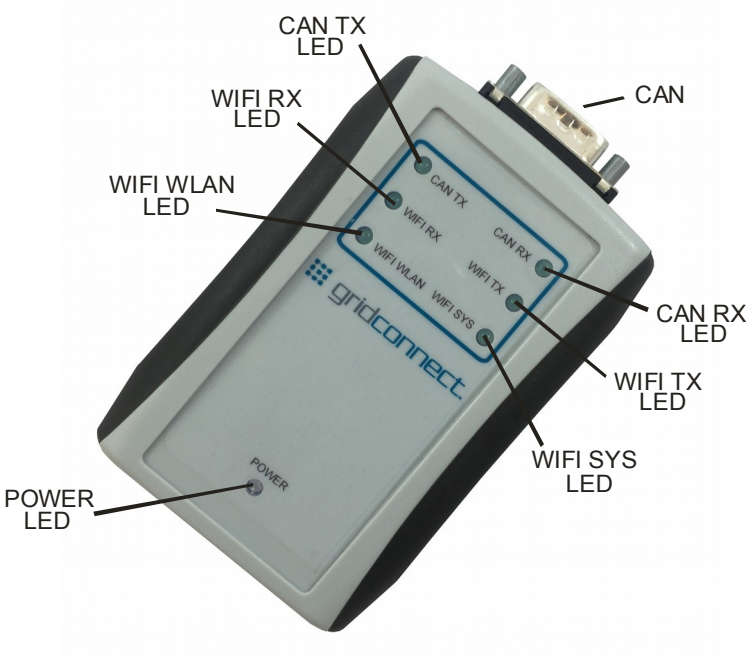
One side is the CAN interface with the COM port interface hard-coded to 921,600N81 with RTS/CTS flow control enabled. The CAN interface can send and receive CAN messages on an ISO11898 CAN network.

The other side consists of the xPICO WiFi module. The configuration for this module is done via an Internet browser. Once the unit has established a connection via its own unique SSID, the browser can be used to login to the xPICO directly and perform configuration.

The CAN side is based on an ARM Cortex-M3 32-bit microcontroller, providing 100 MIPS of processing power with an extremely efficient instructions set.

1.2 Hardware Description

The following drawing shows the location and function of the LEDs and the location of the CAN connector..



1.3 CAN Network

Before trying to send commands to the CAN WiFi, make sure the unit is attached to a valid CAN network with at least one other node attached and running. Without another node on the network, the CAN WiFi will attempt to transmit the message indefinitely (as per CAN 2.0A/B specifications).

Make sure there are terminating resistors on the network. A terminating resistance of 120 ohms is appropriate for ISO11898 drivers. Attempts to communicate over the CAN network without terminating resistors can lead to erratic behavior and many long hours of trouble shooting.

CAN ISO11898 specifies a three wire bus: CAN_H, CAN_L, and GROUND. Failure to provide a common ground between network nodes will create some weird behavior. CAN_H and CAN_L form a differential pair and each one must be referenced to ground in order to work properly.

1.4 Additional Documentation

This manual provides basic information about the CAN WiFi installation and operation. For specific details about configuration and operation, please see the following manuals.

The following documents are available on the product CD.

Title	Description	File Name
CAN-WiFi User Manual	This manual in PDF format.	CAN_WiFi_UG.pdf
CAN WiFi Quick Start Guide	The Quick Start Guide in PDF format.	QuickStart_CAN_WiFi.pdf
xPico WiFi User Guide	xPico WiFi User Guide in PDF format.	xPico-Wi-Fi_UG.pdf

2. Getting Started

2.1 Hardware Installation

2.1.1 Power Options

Power **(+9 to 12VDC)** can be supplied through a through a barrel jack shown below.

2.1.2 Isolated Power Option

The CAN network is isolated for protection of the module. The isolation consists of a DC/DC converter that separates power to the CAN circuits from the rest of the module.

The DC/DC converter is typically used in general purpose power isolation and voltage matching applications, and feature a full industrial operating temperature range of -40°C to +85°C without derating. The device is 2kVDC rated in a UL94V-0 package.

2.1.3 Configuration Push Button

Pressing the recessed push button for **less than 3 seconds** will put the processor into configuration mode.

Pressing the recessed push button for **MORE than 3 seconds** will reset the CAN configuration and the xPico WiFi configuration to factory defaults. The TX/RX LEDs will start blinking in a circle pattern for about 10 seconds.

*WARNING: If you press the config button for **more than 3 seconds**, you will have to re-configure the CAN settings.*

See the Configuration chapter or the Quick Start for detailed operation.

2.1.4 CAN Driver

The CAN driver is a galvanically isolated transceiver that meets or exceeds the specifications of the ISO 11898 standard. The device has the logic input and output buffers separated by a silicon oxide (SiO2) insulation barrier that provides galvanic isolation of up to 5000 VRMS for ISO1050DW and 2500 VRMS for ISO1050DUB and ISO1050LDW. Used in conjunction with isolated power supplies, the device prevents noise currents on a data bus or other circuits from entering the local ground and interfering with or damaging sensitive circuitry.

As a CAN transceiver, the device provides differential transmit capability to the bus and differential receive capability to a CAN controller at signaling rates up to 1 megabit per second (Mbps). Designed for operation in especially harsh environments, the device features cross-wire, overvoltage and loss of ground protection from –27 V to 40 V and over-temperature shut-down, as well as –12 V to 12 V common-mode range.

The ISO1050 is characterized for operation over the ambient temperature range of –55°C to 105°C.

- 2500-VRMS Isolation (ISO1050DUB and ISO1050LDW)
- 5000-VRMS Isolation (ISO1050DW)
- Failsafe Outputs
- Meets or Exceeds ISO 11898 requirements
- Bus-Fault Protection of –27 V to 40 V
- Dominant Time-Out Function
- IEC 60747-5-2 (VDE 0884, Rev. 2) & IEC 61010-1 Approved
- UL 1577 Double Protection Approved
- IEC 60601-1 (Medical) and CSA Approved
- 5 KVRMS Reinforced Insulation per TUV
- Approved for EN/UL/CSA 60950-1 (ISO1050DW)
- Typical 25-Year Life at Rated Working Voltage

DOMINANT TIME-OUT

A dominant time-out circuit in the ISO1050 prevents the driver from blocking network communications if a local controller fault occurs. The time-out circuit is triggered by a falling edge on TXD. If no rising edge occurs on TXD before the time-out of the circuits expires, the driver is disabled to prevent the local node from continuously transmitting a Dominant bit. If a rising edge occurs on TXD, commanding a Recessive bit, the timer will be reset and the driver will be re-enabled. The time-out value is set so that normal CAN communication will not cause the Dominant time-out circuit to expire.

FAILSAFE

If the bus-side power supply Vcc2 is lower than about 2.7V, the power shutdown circuits in the ISO1050 will disable the transceiver to prevent spurious transitions due to an unstable supply. If Vcc1 is still active when this occurs, the receiver output will go to a failsafe HIGH value in about 6 microseconds.

THERMAL SHUTDOWN

The ISO1050 has an internal thermal shutdown circuit that turns off the driver outputs when the internal temperature becomes too high for normal operation. This shutdown circuit prevents catastrophic failure due to short-circuit faults on the bus lines. If the device cools sufficiently after thermal shutdown, it will automatically re-enable, and may again rise in temperature if the bus fault is still present. Prolonged operation with thermal shutdown conditions may affect device reliability.

BUS LOADING

In the CAN standard ISO 11898-2 the driver differential output is specified with a 60Ω load (must be greater than 1.5V) and with a fully-loaded bus (must be greater than 1.2V). The ISO1050 is specified to meet the 1.5V requirement with a 60Ω load, and 1.4V with a 45Ω load. The differential input resistance of the ISO1050 is a minimum of 30KΩ. If the 167 transceivers are in parallel on a bus, this is equivalent to a 180Ω differential load. That transceiver load of 180Ω in parallel with the 60Ω (two 120Ω termination resistors) gives a total 45Ω. Therefore, the ISO1050 supports over 167 transceivers on a single bus segment, with margin to the 1.2V CAN requirement.

2.2 Entering Configuration Mode

See the Quick Start Guide for instructions on how to setup the unit for configuration. The configuration for this module is done via an Internet browser connection to the xPico WiFi module. Once the unit has established a connection via its own unique SSID, the browser can be used to login to the xPICO directly and perform configuration.

Configuration settings for the CAN side are found in this manual. The Quick Start Guide provides information needed to configure the xPico WiFi module for basic tunnel mode operation. Additional xPico WiFi options can be found in the xPico-Wi-Fi_UG.pdf document.

2.2.1 Config Button

Press the Config button (**for less than 3 seconds**). The config prompt should appear as #0#. Type help or ? to display the prompt messages.

```
                                C A N - V C C M - R S 2 3 2
              Application Ver : 02.00A   Bootloader Ver : 01.01B   Hardware Ver : A
                          B R I E F   C O M M A N D   S U M M A R Y
 (00) : get mode
 (01) : set mode vc | cm

 (02) : get com port
 (03) : set com port baud parity bits stop flow

 (04) : get can port
 (05) : set can port baud %sample-point
 (06) : set can port baud %sample-point tol
 (07) : set can port baud %sample-point ?
 (08) : set can port baud %sample-point tol ?

 (09) : set can port (CLK_DIV BRG TSEG1 TSEG2 SJW)

 (10) : get can cm
 (11) : set can cm filter=ON | OFF
 (12) : set can cm mode=NORMAL | MONITOR
 (13) : set can cm ifmt=ASCII | BINARY
 (14) : set can cm ofmt=ASCII | BINARY
 (15) : set can cm eol=CR | LF | CRLF | LFCR | NONE

 (16) : get can vc
 (17) : set can vc tx=id-type id-val
 (18) : set can vc rx=id-type id-val
 (19) : set can vc fs=code
 (20) : set can vc fw=code
 (21) : set can vc ts=timeout
 (22) : set can vc tw=timeout
 (23) : set can vc ww=timeout
 (24) : set can vc wmsg=can-msg

 (25) : get can filter
 (26) : set can filter std id | ext id | std id-min id-max | ext id-min id-max
 (27) : del can filter #
 (28) : del can filter # #
 (29) : del can filter all

 (30) : set can limiter # c=count | # t=time
 (31) : del can limiter #
 (32) : del can limiter all

 (33) : get cfgcmd cm
 (34) : set cfgcmd cm enable=YES | NO

 (35) : get cfgcmd vc
 (36) : set cfgcmd vc enable=YES | NO
 (37) : set cfgcmd vc tbeg=timeout
 (38) : set cfgcmd vc tend=timeout
 (39) : set cfgcmd vc tid=timeout
 (40) : set cfgcmd vc seq="sequence"

 (41) : get timestamp
 (42) : set timestamp enable=YES | NO

 (43) : get can timeout
 (44) : set can timeout can_tx_busy=timeout

 (45) : get sernumcmd cm
 (46) : set sernumcmd cm enable=YES | NO RX=ID | ID RTR TX[0]=ID TX[1]=ID

 (47) : profile #
 (48) : profile copy # #
 (49) : profile note
 (50) : profile note #
 (51) : profile note all
 (52) : profile note="text"
 (53) : profile note #="text"

 (54) : default
 (55) : default #
 (56) : default all

 (57) : get sernum
 (58) : get version

 (59) : test
 (60) : test bps

 (61) : exit
```

```
                                        N O T E S
- View command detail by typing 'help #' where '#' is a number from the above list.

  Ex 1 : help 10

- Most commands can be entered in angle brackets to avoid echo of the individual characters.  When
  angled, no editing or input key echo is performed.  Commands will execute on the '>' terminator.
  No CRLF will be issued and input prompting will be suspened until a CRLF sequence is seem.

  Ex 2 : <set com port 115200 N 8 1 F->

- Normal input has limited line editing capability:

  ESC    : Deletes entire input line.
  BSPACE : Deletes last entered character.

  Note   : In angled mode, editing is disabled.

- Profiles are numbered from 0 to 9.
```

2.3 Mode Options

2.3.1 Get Mode

In the command mode (CM), the CAN WIFI is capable of sending and receiving arbitrary CAN messages via the use of ASCII formatted message strings.

In virtual circuit mode (VC), the CAN WIFI establishes a full-duplex, virtual circuit between itself and another CAN WIFI or application device. By providing a virtual circuit over the CAN network, applications can exchange data in a network-transparent fashion, using existing CAN network cabling as a data link.

```
Command : get mode
Desc : Gets the current operating mode (CM or VC).

Ex 1 : get mode

Resp : <A:CM> or <A:VC>
```

2.3.2 Set Mode

```
Command : set mode
Desc : Sets the current operating mode (CM or VC).

Ex 1 : set mode cm
Ex 2 : set mode vc

Resp : <A:CM> or <A:VC>
```

2.4 CAN Port Options

2.4.1 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. The CAN WIFI provides a means of configuring these parameters so as to support a wide array of network scenarios.

Note that the CAN bit rate settings can be specified as a bit-rate, sampling point, and tolerance. The code will find the best fit solution.

2.4.2 CAN Bus State During Configuration

While the CAN WIFI 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 CAN WIFI while still connected to an active CAN network and not adversely affect network operation.

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

It is the user’s responsibility to ensure that the configuration settings are appropriate for the network to which the CAN WIFI is connected so that the CAN WIFI operates correctly and does not cause erratic network operation.

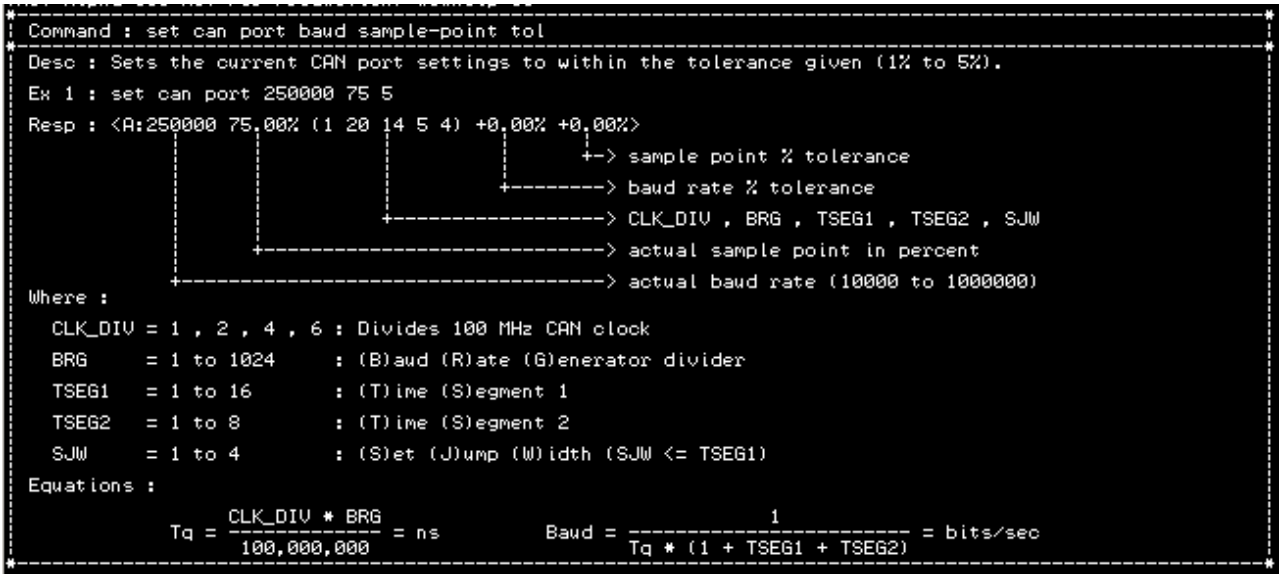
2.4.3 Get CAN Port

```
Command : get can port
Desc : Gets the current CAN port settings.
Ex 1 : get can port
Resp : <A:250000 75.00% (1 20 14 5 4) +0.00% +0.00%>
                                      +-> sample point % tolerance
                                +-----> baud rate % tolerance
                          +-----------> CLK_DIV , BRG , TSEG1 , TSEG2 , SJW
                     +----------------> actual sample point in percent
               +----------------------> actual baud rate (10000 to 1000000)
Where :
  CLK_DIV = 1 , 2 , 4 , 6 : Divides 100 MHz CAN clock
  BRG     = 1 to 1024     : (B)aud (R)ate (G)enerator divider
  TSEG1   = 1 to 16       : (T)ime (S)egment 1
  TSEG2   = 1 to 8        : (T)ime (S)egment 2
  SJW     = 1 to 4        : (S)et (J)ump (W)idth (SJW <= TSEG1)
Equations :
            Tq = CLK_DIV * BRG / 100,000,000 = ns        Baud = 1 / Tq * (1 + TSEG1 + TSEG2) = bits/sec
```

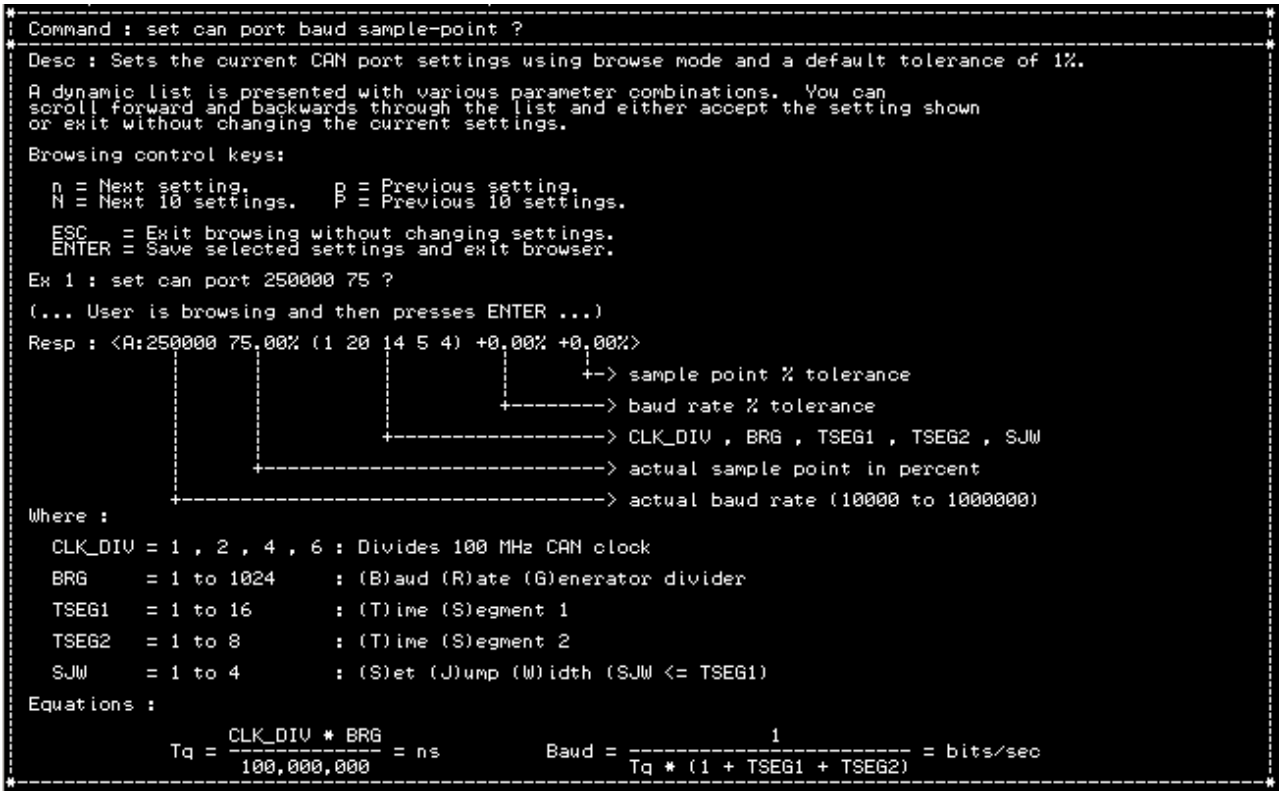
2.4.4 Set CAN Port Baud Sample Point

```
Command : set can port baud sample-point
Desc : Sets the current CAN port settings to within default 1% tolerance.
Ex 1 : set can port 250000 75
Resp : <A:250000 75.00% (1 20 14 5 4) +0.00% +0.00%>
                                      +-> sample point % tolerance
                                +-----> baud rate % tolerance
                          +-----------> CLK_DIV , BRG , TSEG1 , TSEG2 , SJW
                     +----------------> actual sample point in percent
               +----------------------> actual baud rate (10000 to 1000000)
Where :
  CLK_DIV = 1 , 2 , 4 , 6 : Divides 100 MHz CAN clock
  BRG     = 1 to 1024     : (B)aud (R)ate (G)enerator divider
  TSEG1   = 1 to 16       : (T)ime (S)egment 1
  TSEG2   = 1 to 8        : (T)ime (S)egment 2
  SJW     = 1 to 4        : (S)et (J)ump (W)idth (SJW <= TSEG1)
Equations :
            Tq = CLK_DIV * BRG / 100,000,000 = ns        Baud = 1 / Tq * (1 + TSEG1 + TSEG2) = bits/sec
```

2.4.5 Set CAN Port Baud Sample Point Tolerance



2.4.6 Set CAN Port Baud Sample Point Browse



2.4.7 Set CAN Port Baud Sample Point Tolerance

```
Command : set can port baud sample-point tol ?
Desc : Sets the current CAN port settings using browse mode and within the given tolerance (1% to 5%).

A dynamic list is presented with various parameter combinations.  You can
scroll forward and backwards through the list and either accept the setting shown
or exit without changing the current settings.

Browsing control keys:

  n = Next setting.       p = Previous setting.
  N = Next 10 settings.   P = Previous 10 settings.

  ESC   = Exit browsing without changing settings.
  ENTER = Save selected settings and exit browser.

Ex 1 : set can port 250000 75 5 ?

(... User is browsing and then presses ENTER ...)

Resp : <A:250000 75.00% (1 20 14 5 4) +0.00% +0.00%>
                 |      |            |       |      +-> sample point % tolerance
                 |      |            |       +--------> baud rate % tolerance
                 |      |            +----------------> CLK_DIV , BRG , TSEG1 , TSEG2 , SJW
                 |      +-----------------------------> actual sample point in percent
                 +------------------------------------> actual baud rate (10000 to 1000000)
Where :
  CLK_DIV = 1 , 2 , 4 , 6 : Divides 100 MHz CAN clock
  BRG     = 1 to 1024     : (B)aud (R)ate (G)enerator divider
  TSEG1   = 1 to 16       : (T)ime (S)egment 1
  TSEG2   = 1 to 8        : (T)ime (S)egment 2
  SJW     = 1 to 4        : (S)et (J)ump (W)idth (SJW <= TSEG1)
Equations :
                 CLK_DIV * BRG                                  1
            Tq = ------------- = ns         Baud = ------------------------ = bits/sec
                  100,000,000                      Tq * (1 + TSEG1 + TSEG2)
```

2.4.8 Set CAN Port (CLK_DIV BRG TSEG1 TSEG2 SJW)

```
Command : set can port (CLK_DIV BRG TSEG1 TSEG2 SJW)
Desc : Sets the current CAN port settings using exact HW parameters.
Ex 1 : set can port (1 20 14 5 4)
Resp : <A:250000 75.00% (1 20 14 5 4) +0.00% +0.00%>
                 |      |            |       |      +-> sample point % tolerance
                 |      |            |       +--------> baud rate % tolerance
                 |      |            +----------------> CLK_DIV , BRG , TSEG1 , TSEG2 , SJW
                 |      +-----------------------------> actual sample point in percent
                 +------------------------------------> actual baud rate (10000 to 1000000)
Where :
  CLK_DIV = 1 , 2 , 4 , 6 : Divides 100 MHz CAN clock
  BRG     = 1 to 1024     : (B)aud (R)ate (G)enerator divider
  TSEG1   = 1 to 16       : (T)ime (S)egment 1
  TSEG2   = 1 to 8        : (T)ime (S)egment 2
  SJW     = 1 to 4        : (S)et (J)ump (W)idth (SJW <= TSEG1)
Equations :
                 CLK_DIV * BRG                                  1
            Tq = ------------- = ns         Baud = ------------------------ = bits/sec
                  100,000,000                      Tq * (1 + TSEG1 + TSEG2)
```

The ‘TSEG1’ parameter determines the amount of time to wait before the hardware attempts to sample the bit. The value can range from 1 to 16.

The ‘TSEG2’ parameter determines the amount of time remaining before the end of the bit. The value can range from 1 to 8.

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 CAN WIFI can work with other nodes with a large variation is bus oscillator tolerances. The value can range from 1 to 4.

2.5 CAN Command Mode

In the command mode, the CAN WIFI is capable of sending and receiving arbitrary CAN messages via the use of ASCII or Binary formatted message strings (Figure 3).

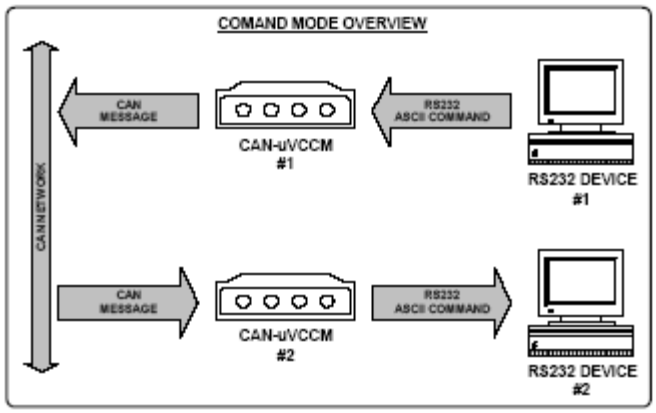


Figure 3

When the CAN WIFI 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 CAN WIFI, it converts it to an ASCII message string and transmits it out of the serial port.

The CAN WIFI supports ten (10) CAN receive message filters consisting of STD, EXT, STD-RANGE, or EXT-RANGE. By using the masks to specify which bits of an identifier are to be compared to the filter value, the CAN WIFI 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 serial 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. See Set CAN Command Mode Message Output Termination on page 19.

2.5.1 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.

Messages starting with '|' will generate a self-receive of the transmitted message. If the message passes filtering, it will be received as if it had been sent from some other node. Terminating with '!' instead of ';' issues a one-shot transmit. Ie. No attempts to automatically retry on error will happen. This is used in time-triggered CAN protocols.

Terminating with '!' instead of ';' issues a one-shot transmit. No attempts to automatically retry on error will happen. This is used in time-triggered CAN protocols.

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.

2.5.2 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 CAN WiFi 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.

2.5.2.1 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.

2.5.3 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 CAN WiFi 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.

2.5.3.1 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.

2.5.4 Appending CR/LF To Received Command Strings

When using the CAN WiFi 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. See Set CAN Command Mode Message Output Termination on page 19.

2.5.5 Binary Formatted Messages

The tables below show the format for Binary message format.

STD-NORMAL Message Format

	BIT-7	BIT-6	BIT-5	BIT-4	BIT-3	BIT-2	BIT-1	BIT-0	
Byte-0	1	1	1	1	1	1	1	1	SYNC
Byte-1	0	0	0	0	0	0	0	0	
Byte-2	EXT=0	RTR=0	ONE-SHOT	SELF-RCEV	LENGTH				TYPE
Byte-3	0	0	0	0	0	ID (10-8)			ID
Byte-4	ID (7-0)								
Byte-5	DATA-0								DATA
Byte-6	DATA-1								
Byte-7	DATA-2								
Byte-8	DATA-3								
Byte-9	DATA-4								
Byte-10	DATA-5								
Byte-11	DATA-6								
Byte-12	DATA-7								

STD-RTR Message Format

	BIT-7	BIT-6	BIT-5	BIT-4	BIT-3	BIT-2	BIT-1	BIT-0	
Byte-0	1	1	1	1	1	1	1	1	SYNC
Byte-1	0	0	0	0	0	0	0	0	
Byte-2	EXT=0	RTR=1	ONE-SHOT	SELF-RCEV	LENGTH				TYPE
Byte-3	0	0	0	0	0	ID (10-8)			ID
Byte-4	ID (7-0)								

EXT-NORMAL Message Format

	BIT-7	BIT-6	BIT-5	BIT-4	BIT-3	BIT-2	BIT-1	BIT-0	
Byte-0	1	1	1	1	1	1	1	1	SYNC
Byte-1	0	0	0	0	0	0	0	0	
Byte-2	EXT=1	RTR=0	ONE-SHOT	SELF-RCEV	LENGTH				TYPE
Byte-3	0	0	0	ID (28-24)					ID
Byte-4	ID (23-16)								
Byte-5	ID (15-8)								
Byte-6	ID (7-0)								
Byte-7	DATA-0								DATA
Byte-8	DATA-1								
Byte-9	DATA-2								
Byte-10	DATA-3								
Byte-11	DATA-4								
Byte-12	DATA-5								
Byte-13	DATA-6								
Byte-14	DATA-7								

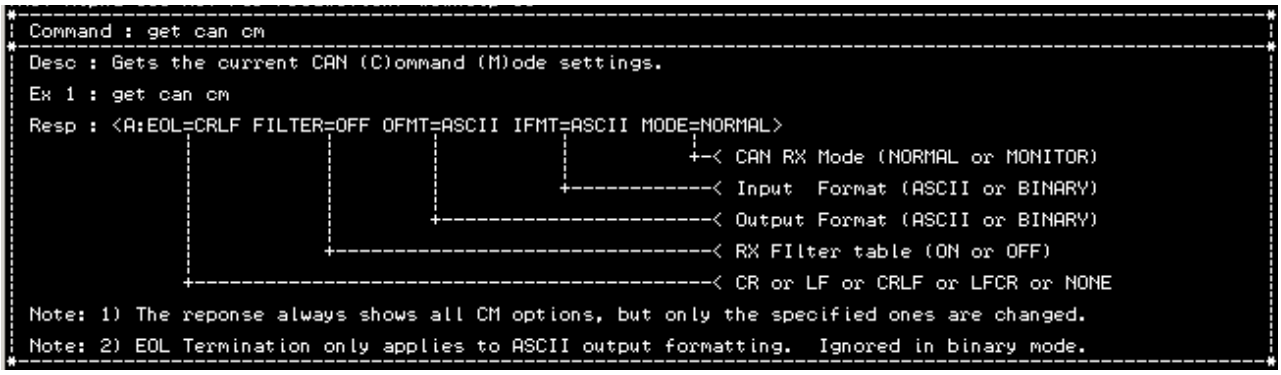
EXT-RTR Message Format

	BIT-7	BIT-6	BIT-5	BIT-4	BIT-3	BIT-2	BIT-1	BIT-0	
Byte-0	1	1	1	1	1	1	1	1	SYNC
Byte-1	0	0	0	0	0	0	0	0	
Byte-2	EXT=1	RTR=1	ONE-SHOT	SELF-RCEV	LENGTH				TYPE
Byte-3	0	0	0	ID (28-24)					ID
Byte-4	ID (23-16)								
Byte-5	ID (15-8)								
Byte-6	ID (7-0)								

DLE Sequences Defined

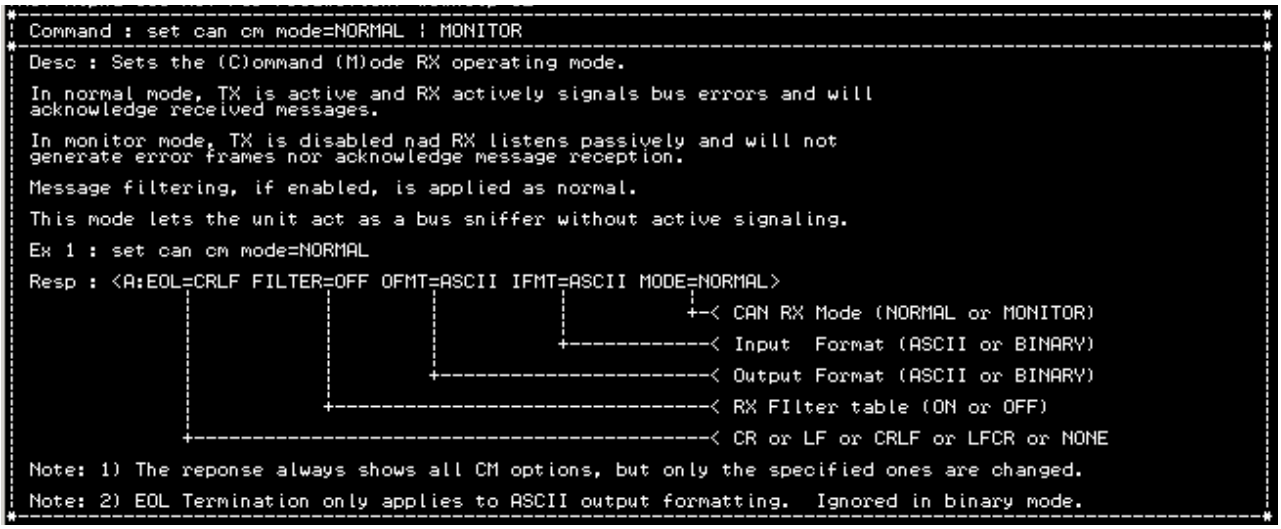
Name	Byte-0	Byte-1	Type	Comment
SYNC	$FF	$00	FRAME	Marks start of message
DATA	$FF	$01	DATA	Defines data value of $FF
DLE RE-SYNC	$FF	$FF	ERROR	Restarts DLE sequence parsing

2.5.6 Get CAN Command Mode Settings

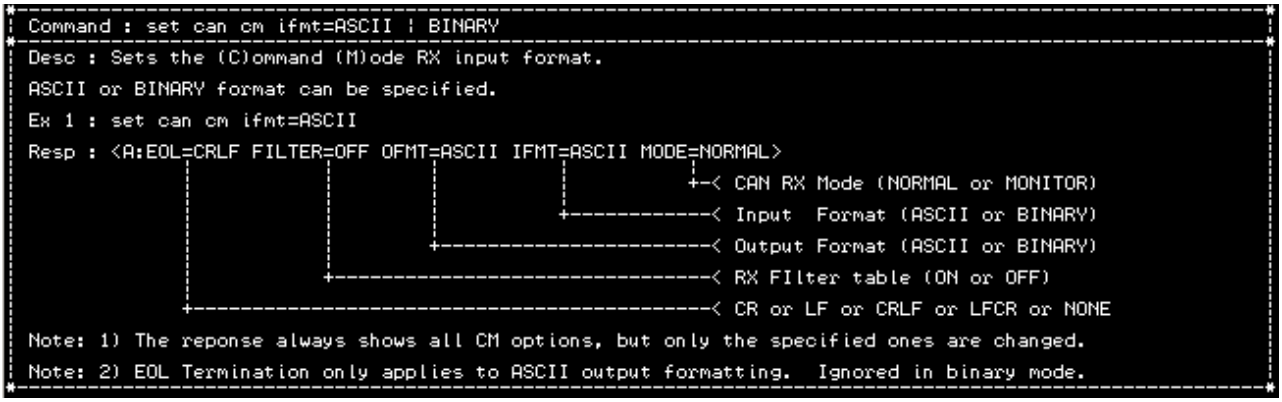


2.5.7 Set CAN Command Mode Filter On/Off

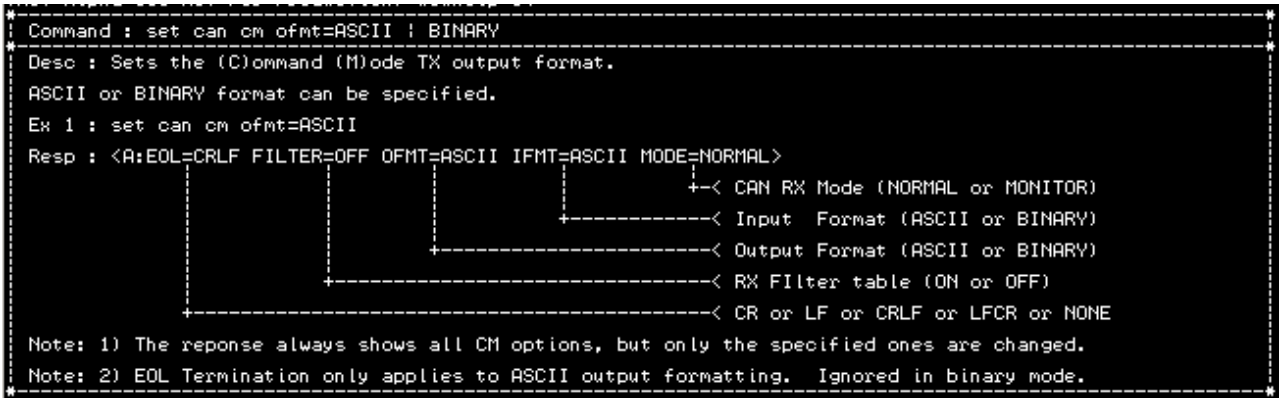
2.5.8 Set CAN Command Mode RX Operating Mode



2.5.9 Set CAN Command Mode RX Input Format



2.5.10 Set CAN Command Mode TX Output Format



2.5.11 Set CAN Command Mode Message Output Termination

```
Command : set can cm eol=CR | LF | CRLF | LFCR | NONE

Desc : Sets the (C)ommand (M)ode message output termination.

If output formatting is in ASCII, then this option determines
how to terminate the message.

Ex 1 : set can cm eol=CRLF

Resp : <A:EOL=CRLF FILTER=OFF OFMT=ASCII IFMT=ASCII MODE=NORMAL>
                                          +-< CAN RX Mode (NORMAL or MONITOR)
                                    +-----< Input  Format (ASCII or BINARY)
                              +-----------< Output Format (ASCII or BINARY)
                        +-----------------< RX FIlter table (ON or OFF)
                  +-----------------------< CR or LF or CRLF or LFCR or NONE

Note: 1) The reponse always shows all CM options, but only the specified ones are changed.

Note: 2) EOL Termination only applies to ASCII output formatting.  Ignored in binary mode.
```

2.6 CAN Virtual Command Mode

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

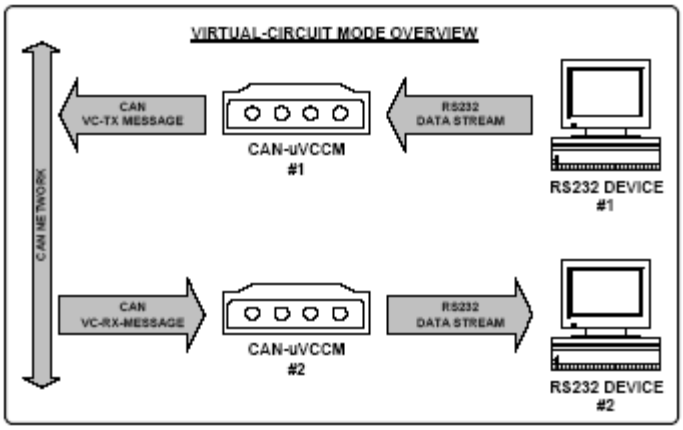


Figure 4

The CAN WiFi takes data bytes coming into the serial port and groups them into CAN messages for transmission. When a remote CAN WiFi or application target receives these messages, it extracts the data bytes and recreates the original data stream on its serial 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 CAN WiFi 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 CAN WIFI provides a timeout feature that will automatically force a transmission of the last accumulated byte(s) after a specified maximum waiting time.

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

It also supports defining and sending of WAKE messages to support CAN networks that go into SLEEP mode. See more detail in help options under ‘set can vc’

2.6.1 The ‘XMIT’ and ‘RCEV’ Identifiers

When configuring the CAN WIFI 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.

2.6.2 ‘XMIT’ Identifier

The ‘XMIT’ identifier is used when the CAN WIFI 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 serial 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 CAN WIFI 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.

2.6.3 ‘RCEV’ Identifier

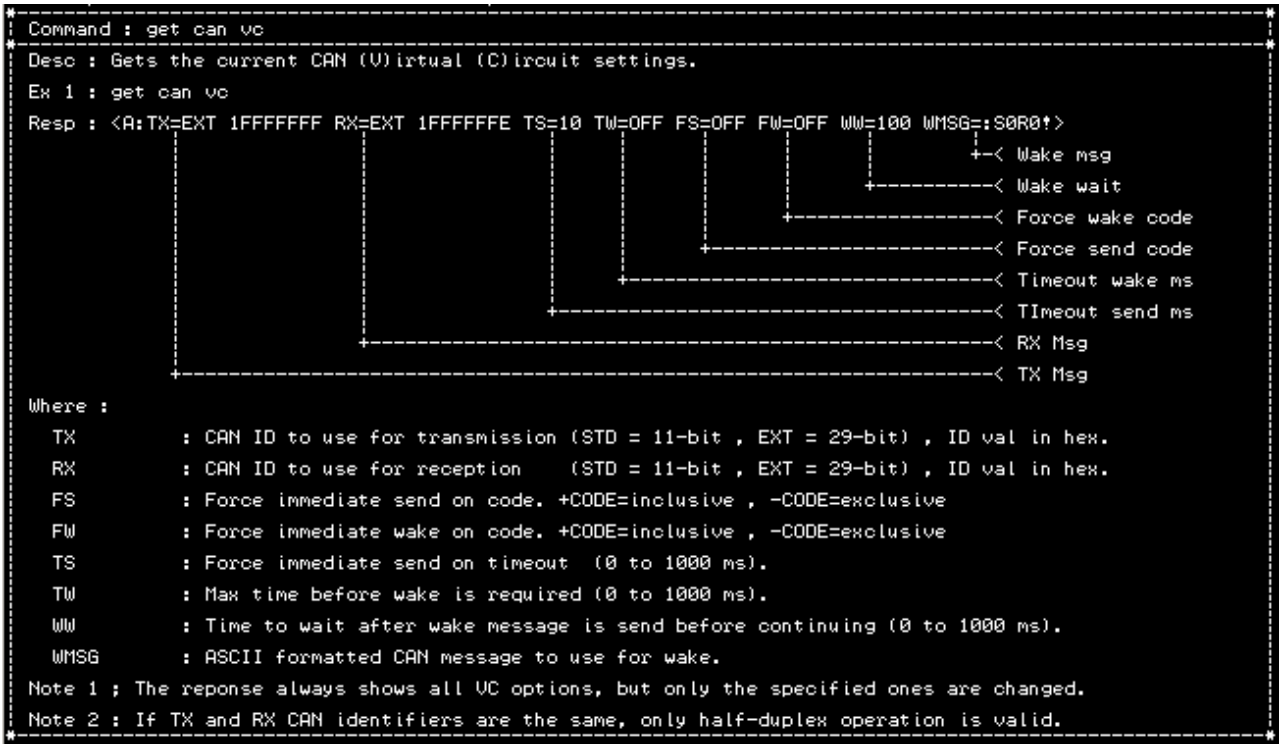
The ‘RCEV’ identifier is used to specify which CAN message should be received by the CAN WIFI when waiting for incoming stream data. The CAN WIFI will discard all other messages.

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

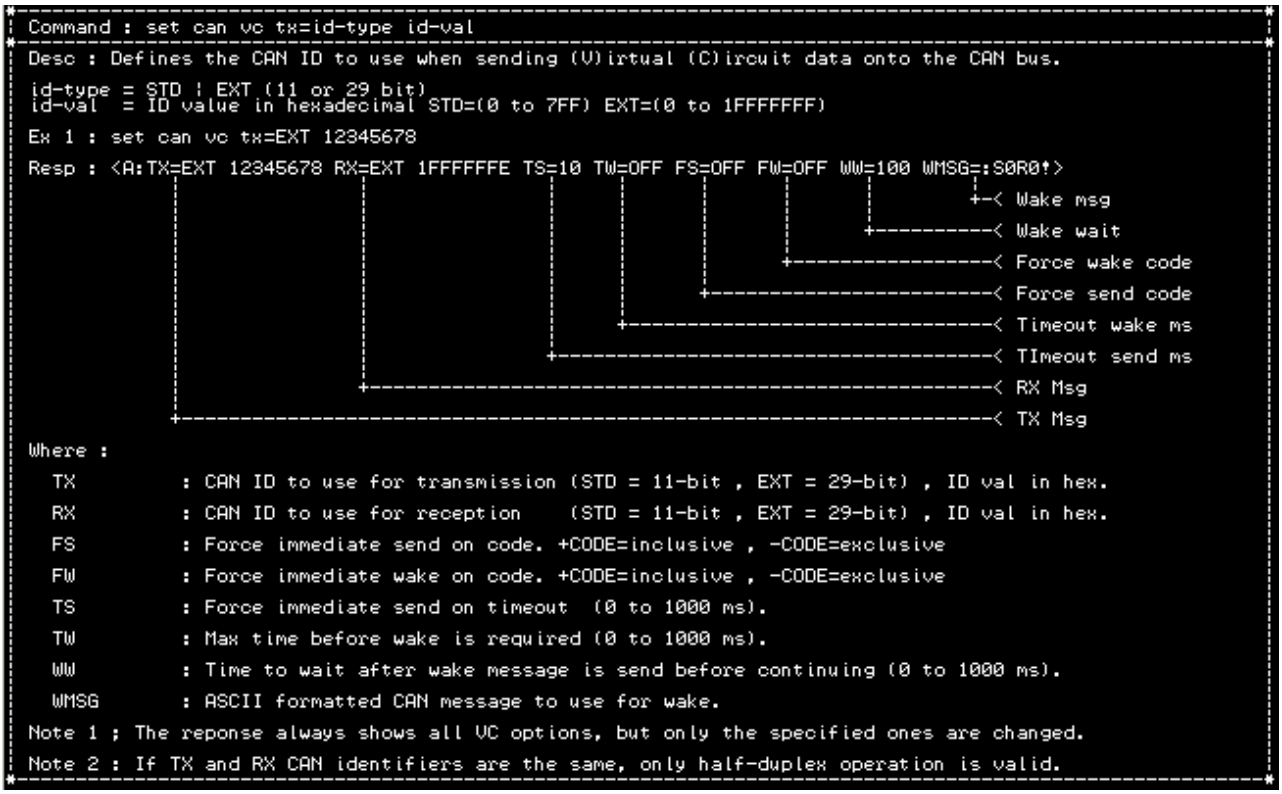
If the received message for any reason contains a length of zero, or is an RTR message, it is discarded and not stream data is generated.

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

2.6.4 Get CAN Virtual Circuit Setting



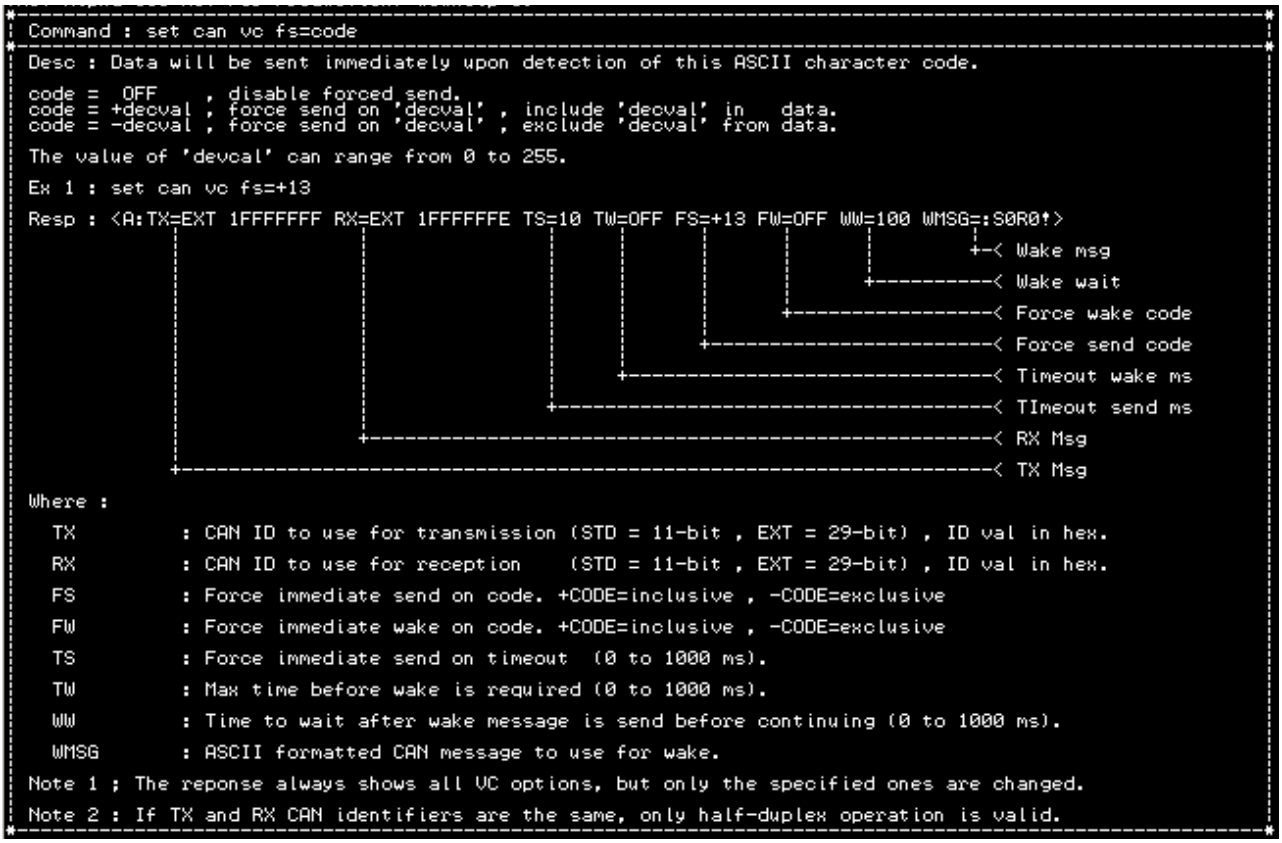
2.6.5 Set CAN Virtual Circuit TX ID



2.6.6 Set CAN Virtual Circuit RX ID

```
Command : set can vc rx=id-type id-val
Desc : Defines the CAN ID to use when receiving (V)irtual (C)ircuit data from the CAN bus.
id-type = STD | EXT (11 or 29 bit)
id-val  = ID value in hexadecimal STD=(0 to 7FF) EXT=(0 to 1FFFFFFF)
Ex 1 : set can vc rx=EXT 12345678
Resp : <A:TX=EXT 1FFFFFFF RX=EXT 12345678 TS=10 TW=OFF FS=OFF FW=OFF WW=100 WMSG=:S0R0↑>
                                                                    +-< Wake msg
                                                          +---------< Wake wait
                                                +-------------------< Force wake code
                                          +-----------------------------< Force send code
                                    +-------------------------------------< Timeout wake ms
                              +-----------------------------------------------< TImeout send ms
                        +-----------------------------------------------------< RX Msg
              +-------------------------------------------------------------------< TX Msg
Where :
  TX          : CAN ID to use for transmission (STD = 11-bit , EXT = 29-bit) , ID val in hex.
  RX          : CAN ID to use for reception    (STD = 11-bit , EXT = 29-bit) , ID val in hex.
  FS          : Force immediate send on code. +CODE=inclusive , -CODE=exclusive
  FW          : Force immediate wake on code. +CODE=inclusive , -CODE=exclusive
  TS          : Force immediate send on timeout  (0 to 1000 ms).
  TW          : Max time before wake is required (0 to 1000 ms).
  WW          : Time to wait after wake message is send before continuing (0 to 1000 ms).
  WMSG        : ASCII formatted CAN message to use for wake.
Note 1 ; The reponse always shows all VC options, but only the specified ones are changed.
Note 2 : If TX and RX CAN identifiers are the same, only half-duplex operation is valid.
```

2.6.7 Set CAN Virtual Circuit Forced Send Code

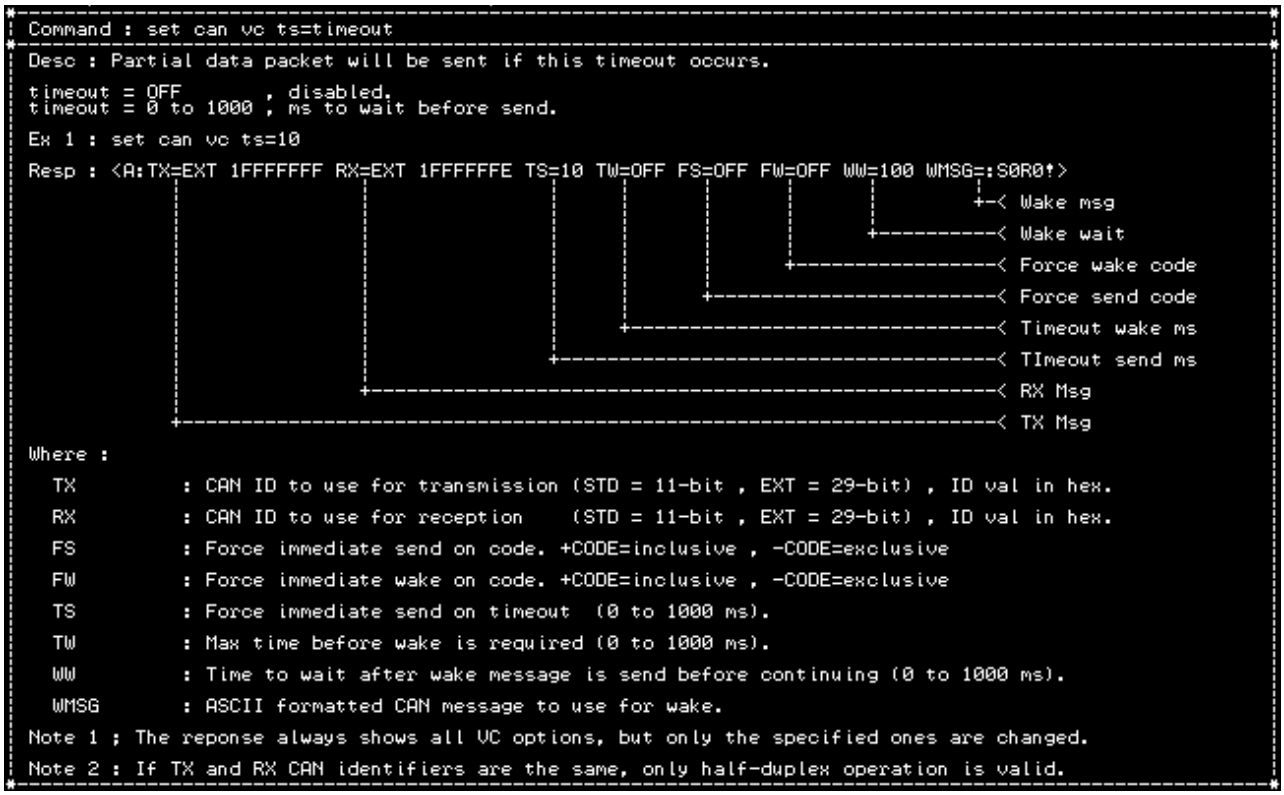


```
Command : set can vc fs=code

Desc : Data will be sent immediately upon detection of this ASCII character code.

code =  OFF    , disable forced send.
code = +decval , force send on 'decval' , include 'decval' in   data.
code = -decval , force send on 'decval' , exclude 'decval' from data.

The value of 'devcal' can range from 0 to 255.

Ex 1 : set can vc fs=+13

Resp : <A:TX=EXT 1FFFFFFF RX=EXT 1FFFFFFE TS=10 TW=OFF FS=+13 FW=OFF WW=100 WMSG=:S0R0!>
                                                                        +-< Wake msg
                                                                  +-----------< Wake wait
                                                            +-----------------< Force wake code
                                                      +-------------------------< Force send code
                                                +---------------------------------< Timeout wake ms
                                          +-------------------------------------< TImeout send ms
                                    +-----------------------------------------------< RX Msg
                              +-------------------------------------------------------< TX Msg

Where :
  TX         : CAN ID to use for transmission (STD = 11-bit , EXT = 29-bit) , ID val in hex.
  RX         : CAN ID to use for reception    (STD = 11-bit , EXT = 29-bit) , ID val in hex.
  FS         : Force immediate send on code. +CODE=inclusive , -CODE=exclusive
  FW         : Force immediate wake on code. +CODE=inclusive , -CODE=exclusive
  TS         : Force immediate send on timeout  (0 to 1000 ms).
  TW         : Max time before wake is required (0 to 1000 ms).
  WW         : Time to wait after wake message is send before continuing (0 to 1000 ms).
  WMSG       : ASCII formatted CAN message to use for wake.
Note 1 ; The reponse always shows all VC options, but only the specified ones are changed.
Note 2 : If TX and RX CAN identifiers are the same, only half-duplex operation is valid.
```

2.6.8 Set CAN Virtual Circuit Forced Wake Code

```
Command : set can vc fw=code

Desc : Wake message will be sent immediately upon detection of this ASCII character code.

code =  OFF    , disable forced wake.
code = +decval , force wake on 'decval' , include 'decval' in   data.
code = -decval , force wake on 'decval' , exclude 'decval' from data.

The value of 'devcal' can range from 0 to 255.

Ex 1 : set can vc fw=-13

Resp : <A:TX=EXT 1FFFFFFF RX=EXT 1FFFFFFE TS=10 TW=OFF FS=OFF FW=-13 WW=100 WMSG=:S0R0!>
                                                                    +-< Wake msg
                                                          +---------< Wake wait
                                                +-------------------< Force wake code
                                          +-------------------------< Force send code
                                    +-------------------------------< Timeout wake ms
                              +-------------------------------------< TImeout send ms
                        +-----------------------------------------------< RX Msg
                  +-------------------------------------------------------< TX Msg

Where :
  TX         : CAN ID to use for transmission (STD = 11-bit , EXT = 29-bit) , ID val in hex.
  RX         : CAN ID to use for reception    (STD = 11-bit , EXT = 29-bit) , ID val in hex.
  FS         : Force immediate send on code. +CODE=inclusive , -CODE=exclusive
  FW         : Force immediate wake on code. +CODE=inclusive , -CODE=exclusive
  TS         : Force immediate send on timeout  (0 to 1000 ms).
  TW         : Max time before wake is required (0 to 1000 ms).
  WW         : Time to wait after wake message is send before continuing (0 to 1000 ms).
  WMSG       : ASCII formatted CAN message to use for wake.
Note 1 ; The reponse always shows all VC options, but only the specified ones are changed.
Note 2 : If TX and RX CAN identifiers are the same, only half-duplex operation is valid.
```

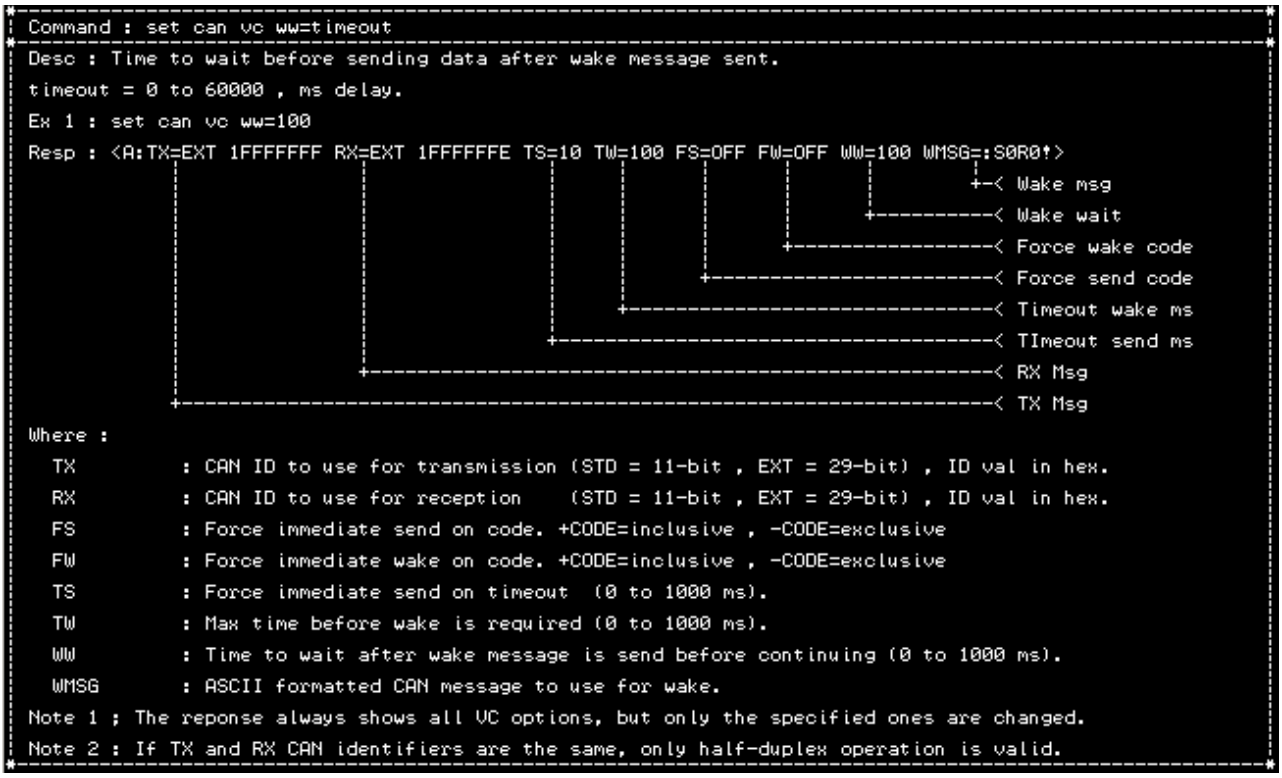
2.6.9 Set CAN Virtual Circuit Timeout Send



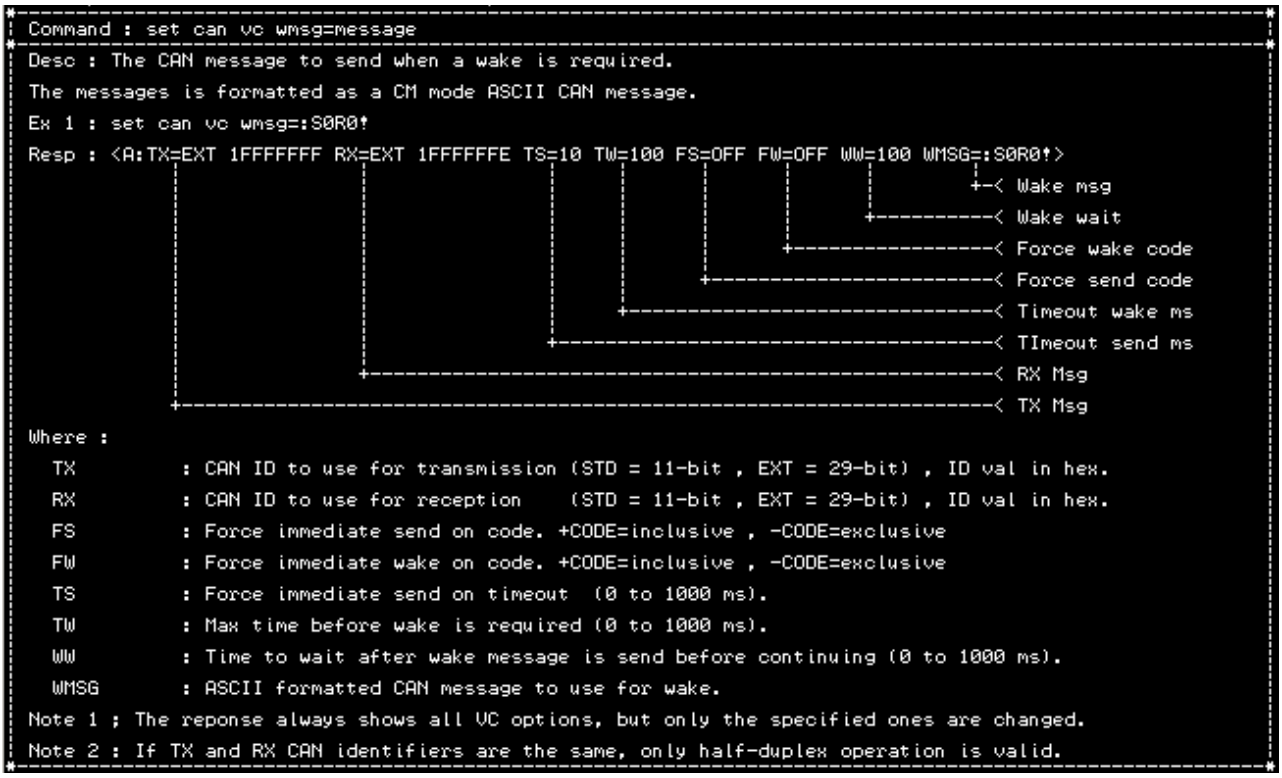
2.6.10 Set CAN Virtual Circuit Wake Timeout

```
Command : set can vc tw=timeout
Desc : Maximum CAN bus idle time before wake is required.

timeout = OFF       , disabled.
timeout = 0 to 60000 , ms window.

Ex 1 : set can vc tw=100

Resp : <A:TX=EXT 1FFFFFFF RX=EXT 1FFFFFFE TS=10 TW=100 FS=OFF FW=OFF WW=100 WMSG=:S0R0↑>
                                                                    +-< Wake msg
                                                                +---< Wake wait
                                                            +-------< Force wake code
                                                        +-----------< Force send code
                                                    +---------------< Timeout wake ms
                                                +-------------------< TImeout send ms
                                            +-----------------------< RX Msg
                                        +---------------------------< TX Msg

Where :
  TX         : CAN ID to use for transmission (STD = 11-bit , EXT = 29-bit) , ID val in hex.
  RX         : CAN ID to use for reception    (STD = 11-bit , EXT = 29-bit) , ID val in hex.
  FS         : Force immediate send on code. +CODE=inclusive , -CODE=exclusive
  FW         : Force immediate wake on code. +CODE=inclusive , -CODE=exclusive
  TS         : Force immediate send on timeout  (0 to 1000 ms).
  TW         : Max time before wake is required (0 to 1000 ms).
  WW         : Time to wait after wake message is send before continuing (0 to 1000 ms).
  WMSG       : ASCII formatted CAN message to use for wake.
Note 1 ; The reponse always shows all VC options, but only the specified ones are changed.
Note 2 : If TX and RX CAN identifiers are the same, only half-duplex operation is valid.
```

2.6.11 Set CAN Virtual Circuit Wait after Wakeup



2.6.12 Set CAN Virtual Circuit Wakeup Message



When in VC mode and enabling force-wake and/or force-tx codes, these codes will be overidden by the CONFIG sequence check. For example, if either of the force-wake or force-tx codes is enabled and the CONFIG command is enabled, should the config command sequence match one of the force codes, it will be treated as a config command character.

To avoid this, either change the force codes or change the config command sequence.

2.7 Defining Filter Entries

- CAN messages can be filtered according to a message filter list.
- The list defines which identifiers will be received and which will be ignored.
- Definitions can be either a single ID or an ID range.
- Standard (11-bit) and Extended (29-bit) IDs can be defined.
- Definitions must not conflict nor overlap.
- ID values are in hexadecimal and do not need a '$' or '0x' prefix.
- The list can be enabled or disabled.
- When disabled, all CAN messages are received regardless of the definitions in the list.
- When enabled, only messages which match filter definitions in the list will be passed through.
- An empty filter list that is enabled will receive nothing, but still permit transmission (making the unit effectively TX only).
- Multiple filter definitions can be specified in a single command line.
- The maximum number of filter definitions possible is ten (10) per profile.

EX-1: set can filter std 100 ext 200 std 1 10 ext 300 3ff

This will define four filters as follows:

- 11-bit : { 0x100 , 0x001 to 0x010 }
- 29-bit : (0x200 , 0x300 to 0x3FF }

All other IDs received are ignored.

2.8 Using Message Filters

In command mode, the CAN WIFI supports message filtering through the use of 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 data bandwidth on a busy network by only allowing certain messages to be received.

To use the filters, they must be enabled, their type must be specified, and appropriate ID values must be assigned. This configuration is done while in the configuration mode.

2.8.1 How Filtering Works

Whenever the CAN WIFI 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 serial port. If any filters are enabled, the CAN WIFI 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 serial port. If no matches were successful, the message is discarded.

Note: The output format depends on mode : ASCII or BINARY

2.8.2 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.

A filter is configured in the following order:

1) Enable the filter

2) Specify if standard or extended and if single ID or ID range.

Once all configuration parameters have been saved and the CAN WIFI enters the normal operating mode, the new filter settings will become active and will be applied to each received CAN message.

2.8.3 Get CAN Filter

- The list can be viewed with the ‘get can filter’ command.
- The list is sorted in ascending order.
- Each entry is assigned an index number starting from zero.
- The index number is used to identify an entry for deletion.

```
Command : get can filter
Desc : Returns a list of defined CAN filters.

A numbered list of filter definitions is returned.
If the command was angled, the list is returned on one line and delimited by angle brackets.
Otherwise each filter definition is shown on a separate line to make it easier to read.

Ex 1 : get can filter

Resp : <A:NONE>

Ex 2 : get can filter

Resp :
<A:
#0=STD 000
#1=STD 001 0FF
#2=EXT 00000000
#3=EXT 00000001 000000FF
>

Ex 3 : <get can filter>

Resp : <A:#0=STD 000 #1=STD 001 0FF #2=EXT 00000000 #3=EXT 00000001 000000FF>

Note 1 : All identifier values are in hexadecimal.

Note 2 : The decimal number between the '#' and '=' is the index into the filter list.
```

2.8.4 Set CAN Filter

Format: set can filter std id xx

```
                        |   |  |____ filter bits
                        |   |
                        |   |___ CAN ID or (id-min id-max)
                        |
                        |____ std or ext
```

Note: Once you set the CAN filter, make sure you set the command mode Filter to ON.

```
Command : set can filter
Desc : Defines a new CAN filter definition.

Will create a new filter and add it to the filter table.  If the new filter conflicts
with any existing fitlers, an error response is returned and the filter is discarded.

Ex 1 : set can filter std 0 std 1 ff ext 0 ext 1 ff

Resp :
<A:
#0=STD 000
#1=STD 001 0FF
#2=EXT 00000000
#3=EXT 00000001 000000FF
>

Note 1 : All identifier values are in hexadecimal.

Note 2 : The decimal number between the '#' and '=' is the index into the filter list.

Note 3 : Multiple filter definitions can be defined in a single command (see above example).
```

2.8.5 Delete CAN Filter

- Either a single entry or and entry range can be specified.
- Entries are identified by their index number.
- Indexes remaining are re-sorted so there are no gaps.

EX-1: del can filter 1

```
Command : del can filter #
Desc : Deletes a CAN filter definition from the filter table.

The decimal index given identifies the filter to be deleted.
If the index is out of range or the table is empty, an error is returned.

The filter identified by the index is deleted and all filters after that are
renumbered using the given index as the starting point.

Ex 1 : del can filter 1

Before :
<A:
#0=STD 000
#1=STD 001 0FF
#2=EXT 00000000
#3=EXT 00000001 000000FF
>

After :
<A:
#0=STD 000
#1=EXT 00000000
#2=EXT 00000001 000000FF
>

Note 1 : All identifier values are in hexadecimal.

Note 2 : The decimal number between the '#' and '=' is the index into the filter list.
```

2.8.6 Delete CAN Filter Range

```
Command : del can filter # #
Desc : Deletes a CAN filter definition range from the filter table.

The decimal indices given identify the range of filters to be deleted.
The first index is the starting point and the second index is the end point.

The filters defined in the range are delted and all filters after that are
renumbered using the first index as the starting point.

If either index is out of range of the table, then an error is returned.

Ex 1 : del can filter 1 2

Before :
<A:
#0=STD 000
#1=STD 001 0FF
#2=EXT 00000000
#3=EXT 00000001 000000FF
>

After :
<A:
#0=STD 000
#1=EXT 00000001 000000FF
>

Note 1 : All identifier values are in hexadecimal.

Note 2 : The decimal number between the '#' and '=' is the index into the filter list.
```

2.8.7 Delete CAN Filter All

```
Command : del can filter all
Desc : Deletes all CAN filter definitions.
Ex 1 : del can filter all
Resp : <A:All filters deleted>
```

2.9 CAN Message Reception Rate Limiting

It is now possible to specify either a skip count or timeout between consecutive CAN messages received. This allows the CAN tool to effectively reduce the rate of CAN message receptions without having to make changes in either the CAN message source or the CAN message sink application software.

There are two methods that can be specified:

Count: Specifies the number of consecutive CAN messages to skip before allowing a single CAN message to pass

(Ex: for a count of 2 : MSG , SKIP , SKIP , MSG , SKIP , SKIP, ….).

Time: Specifies the minimum number of milliseconds between two consecutive CAN messages. A message will be allowed to pass and the internal timer will be reset. Any other messages that arrive will not be passed until the timer has counted down to zero. Upon timeout, the next received message will be passed and the cycle repeats.

The count method is used to divide the frequency of incoming messages while the time method is used to define a maximum CAN message rate independent of incoming frequency.

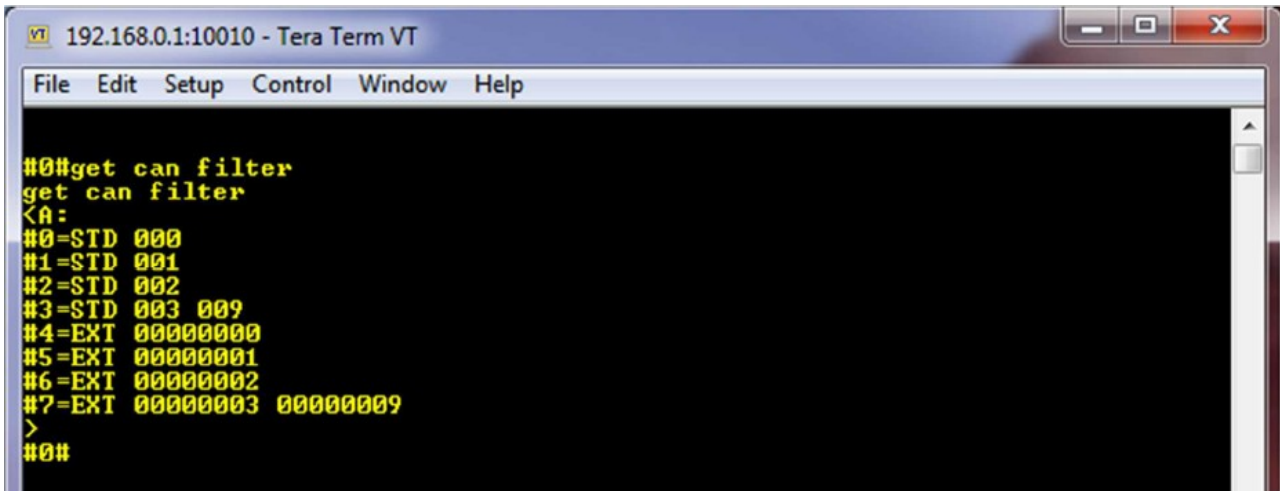
Limiters are defined by attaching them to existing CAN RX filters. The CAN tool supports the definition of up to ten distinct CAN message filter definitions. A CAN filter is first specified to define a CAN identifier (either STD or EXT) and then a limiter can be defined and attached to the CAN filter. Only single CAN ID filter types can be specified for limiting. Filter ID ranges cannot be assigned limiters for all possible definitions, hence they are not supported.

Limiters, once defined, can be deleted without affecting the filter definition.

Once all filters have been defined, their respective limiters attached, filtering enabled, and configuration mode exited, the CAN tool will begin applying the filter & limiter rules to incoming messages.

In order to maintain backwards compatibility with earlier firmware versions, the response to a filter get command remains unchanged if limiters are not defined. Once a limiter is defined for a filter, the response is modified to append a short descriptor indicating the type and value (ie: time or count). This ensures 100% backwards compatibility with applications that do not use the limiter feature.

The following command descriptions and examples assume that CAN filters have already been defined as follows:

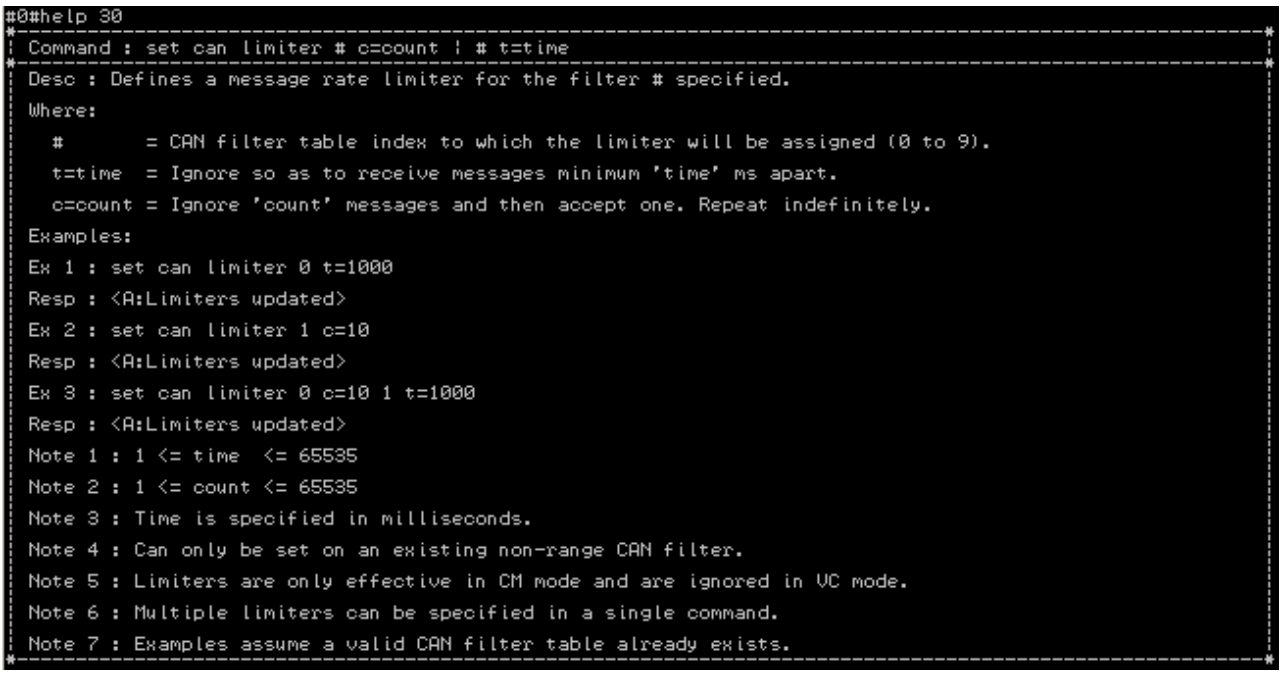


From the example filter configuration above, entries { 0, 1, 2, 4, 5, 6 } are all single ID specifiers and as such, can have limiters assigned. The other entries define ranges and are not eligible to have limiters attached.

Note that in addition to the filters being defined, they must also be enabled.

The following pages will show the command syntax along with some examples and screenshots of what to expect as responses.

2.9.1 Set CAN Limiter # C=Count | T=Time



The value for '#' represents the numeric filter ID as shown in the screenshot above.

'COUNT': 1 <= COUNT <= 65535: The skip count.

'TIME': 1 <= TIME <= 65535: The minimum amount of time (in ms) between consecutive messages.

EXAMPLE #1: DEFINE A LIMITER OF 1 SECOND FOR FILTER ID #0

```
#0#set can liniter 0 t=1000
<A:Limiters updated>
#0#get can filter
<A:
#0=STD 000 T=01000
#1=STD 001
#2=STD 002
#3=STD 003 009
#4=EXT 00000000
#5=EXT 00000001
#6=EXT 00000002
#7=EXT 00000003 00000009
>
#0#_
```

EXAMPLE #2: DEFINE A LIMITER TO PASS EVERY TENTH MESSAGE FOR FILTER ID #1

```
#0#set can liniter 1 c=9
<A:Limiters updated>
#0#get can filter
<A:
#0=STD 000 T=01000
#1=STD 001 C=00009
#2=STD 002
#3=STD 003 009
#4=EXT 00000000
#5=EXT 00000001
#6=EXT 00000002
#7=EXT 00000003 00000009
>
#0#_
```

Note: A value of nine (9) was specified to create an effective divider of ten (10). As ‘COUNT’ represents a skip count, we wish to skip nine messages and pass the tenth. This effectively creates a rate divider of ten.

EXAMPLE #3: DEFINE TWO LIMITERS AT THE SAME TIME FOR FILTER ID # 5 AND # 6

```
#0#set can liniter 5 c=10 6 t=5000
<A:Limiters updated>
#0#get can filter
<A:
#0=STD 000 T=01000
#1=STD 001 C=00009
#2=STD 002
#3=STD 003 009
#4=EXT 00000000
#5=EXT 00000001 C=00010
#6=EXT 00000002 T=05000
#7=EXT 00000003 00000009
>
#0#_
```

Note: Multiple limiters can be specified in a single command by separating their respective definitions with a space.

2.9.2 Del CAN Limiter # | All

The value for ‘#’ represents the numeric filter ID.

If ‘all’ is specified on its own, then all existing limiters will be deleted.

This command will remove a previously defined limiter from a CAN filter. The CAN filter will continue to exist but without a limiter.

EXAMPLE #4: DELETE LIMITER ON FILTER #0

```
#0#del can limiter 0
<A:Limiters deleted>
#0#get can filter
<A:
#0=STD 000
#1=STD 001 C=00009
#2=STD 002
#3=STD 003 009
#4=EXT 00000000
#5=EXT 00000001 C=00010
#6=EXT 00000002 T=05000
#7=EXT 00000003 00000009
>
#0#_
```

Note: Multiple limiters can be deleted in a single command by separating their respective IDs with a space.

EXAMPLE #5: DELETE ALL DEFINED LIMITERS

```
#0#del can limiter all
<A:All limiters deleted>
#0#get can filter
<A:
#0=STD 000
#1=STD 001
#2=STD 002
#3=STD 003 009
#4=EXT 00000000
#5=EXT 00000001
#6=EXT 00000002
#7=EXT 00000003 00000009
>
#0#_
```

2.10 Config Entry Commands

It is now possible to enter configuration mode remotely through the COM port in either CM or VC mode.

In CM mode: ASCII mode command = :CONFIG;

BINARY mode command = FF 00 FF 02 43 4F 4E 46 49 47

Note: The last six bytes of the BINARY command are ASCII 'C' 'O' 'N' 'F' 'I' 'G'

In VC mode: 1-second idle time, followed by three '!' characters, followed by 1-second idle time.

The exact sequence can be specified by the user.

The 'set/get cfgcmd' configuration command will allow you to enable/disable the CONFIG command and set additional parameters described below.

CM Mode:

get cfgcmd cm : Returns the current state of the CM mode config command

set cfgcmd cm enabled=yes | no : Enables or disables the command in CM mode (default is enabled)

VC Mode:

get cfgcmd vc: Returns the current state of the VC mode config command

set cfgcmd vc enabled=yes | now: Enables or disables the command in VC mode (default is disabled)

set cfgcmd vc tbeg=timeout: Minimum IDLE time to wait before matching "!!!" sequence
: 1000 to 10000 ms (default 1000)

set cfgcmd vc tend=timeout: Minimum IDLE time to wait after matching "!!!" sequence
: 1000 to 10000 ms (default 1000)

set cfgcmd vc tid=timeout: Maximum IDLE time between sequence characters
: 10 to 1000 ms (default 1000)

set cfgcmd vc seq="text": Defines the command sequence text to match
: 1 to 10 characters long (default "!!!")

Note: For the text specified in the 'seq' parameter, arbitrary ASCII values can be entered with the backslash escape sequence followed by three decimal digits from 0 to 255.

Common escape sequences are defined:

\r : CR

\n : LF

\t : TAB

\" : " (double quote)

\\ : BACKSLASH

Ex 1 : "\000" : Binary 0x00

Ex 2 : "\r\n" " CR followed by LF

2.10.1 Get cfgcmd cm

```
Command : get cfgcmd cm
Desc : Gets the current CM-Mode config command settings.
Ex 1 : get cfgcmd cm
Resp : <A:enabled=yes>
```

2.10.2 Set cfgcmd cm enable=YES | NO

```
Command : set cfgcmd cm enabled=YES ! NO
Desc : Enables or disables CM-Mode config command.
Ex 1 : set cfgcmd cm enabled=no
Resp : <A:enabled=no>
```

2.10.3 Get cfgcmd vc

```
Command : get cfgcmd vc
Desc : Gets the current VC-Mode config command settings.
Ex 1 : get cfgcmd vc
Resp : <A:enabled=no tbeg=1000 tend=1000 tid=1000 seq="!!!">
```

2.10.4 set cfgcmd vc enable=YES | NO

```
Command : set cfgcmd vc enabled=YES | NO
Desc : Enables or disables VC-Mode config command.
Ex 1 : set cfgcmd vc enabled=no
Resp : <A:enabled=no tbeg=1000 tend=1000 tid=1000 seq="!!!">
Note: 1) Response always includes all parameters.
```

2.10.5 set cfgcmd vc tbeg=timeout

```
Command : set cfgcmd vc tbeg=timeout
Desc : Sets min idle time (ms) before VC-Mode config command can be recognized.
Ex 1 : set cfgcmd vc tbeg=1000
Resp : <A:enabled=no tbeg=1000 tend=1000 tid=1000 seq="!!!">
Note: 1) Response always includes all parameters.
Note: 2) Timeout is from 1000 to 10000 ms.
```

2.10.6 set cfgcmd vc tend=timeout

```
Command : set cfgcmd vc tend=timeout
Desc : Sets min idle time (ms) after VC-Mode config command can be recognized.
Ex 1 : set cfgcmd vc tend=1000
Resp : <A:enabled=no tbeg=1000 tend=1000 tid=1000 seq="!!!">
Note: 1) Response always includes all parameters.
Note: 2) Timeout is from 1000 to 10000 ms.
```

2.10.7 set cfgcmd vc tid=timeout

```
Command : set cfgcmd vc tid=timeout
Desc : Sets max time (ms) allowed between VC-Mode config command sequence bytes.
Ex 1 : set cfgcmd vc tid=1000
Resp : <A:enabled=no tbeg=1000 tend=1000 tid=1000 seq="!!!">
Note: 1) Response always includes all parameters.
Note: 2) Timeout is from 10 to 1000 ms.
```

2.10.8 set cfgcmd vc seq=”sequence”

```
Command : set cfgcmd vc seq="text"
Desc : Sets the VC-Mode config command sequence bytes.
Ex 1 : set cfgcmd vc seq="!!!"
Resp : <A:enabled=no tbeg=1000 tend=1000 tid=1000 seq="!!!">
Note: 1) Response always includes all parameters.
Note: 2) Use "\nnn" to enter 3-digit decimal ASCII values (000 to 255).
Note: 3) For '<' , '>' , and '"' : Use ASCII values.
Note: 4) Use \t for TAB, \r for CR , \n for LF , \" for QUOTE , \\ for BACK-SLASH
Note: 5) Command sequence length constraints : 1 <= length <= 10
```

2.11 CAN Timestamp Commands

Received CAN messages can now have a 16-bit time-stamp appended to them with a 1-ms resolution. For backward compatibility, this feature is disabled by default and can be enabled by a configuration command.

In configuration mode:

To enable: set timestamp enable=yes

To disable: set timestamp enable=no

To determine the current state of timestamp enable: get timestamp

This will return a string with the enable status: Ex: <A:enabled=yes>

Note: commands are not case sensitive.

CAN RX Message Format Changes (when timestamping is enabled):

ASCII Mode:

In ASCII mode, the timestamp is appended to the data block in uppercase HEX ASCII in the same way that the data field is presented (i.e.: two ASCII HEX digits to represent a byte). The '@' character is inserted between the last DATA digit and the first TIMESTAMP digit to differentiate between the two fields.

Zero-length messages appear as they would normally, except the prefixed timestamp is added.

Ex 1:S12N12@F00F: This is a 11-bit identifier with 1 data byte and timestamp of $F00F ms.

Ex 2:X13N@2EDF: This is a 29-bit identifier with 0 data bytes and timestamp of $2EDF ms.

Ex 3:S14R5@15E5: This is a 11-bit identifier with RTR=5 and timestamp of $15E5 ms.

In short, the timestamp is always prefixed with a '@' character.

The timestamp is variable length from 1 digit to 4 digits (uppercase ASCII HEX).

BINARY Mode:

In binary mode, the received CAN messages are extended by appending the time-stamp as two binary bytes (MSB first).

The encoding of the two byte is the same as for the DATA field (i.e., If any of the time-stamp bytes is a synchronization character, it is escaped).

The time-stamp field is always two logical bytes, but can be expanded up to four in the case where both MSB and LSB bytes are synchronization characters.

2.11.1 get timestamp

```
Command : get timestamp
Desc : Gets the current timestamp settings.
Ex 1 : get timestamp
Resp : <A:enabled=no>
Note 1 : This option is only effective in CM mode and is ignored in VC mode.
Note 2 : In ASCII  : Variable length hex value just after the data field with an '@' prefix.
Note 3 : In BINARY : Appended as a two-byte big-endian value to a normal message.
Note 4 : Time is measured in millisecond resolution.
Note 5 : A timestamp is only appended (if enabled) to received CAN messages.
```

2.11.2 set timestamp

```
Command : set timestamp enable=yes | no
Desc : Sets the timestamp enable/disable option.
Ex 1 : set timestamp enable=no
Resp : <A:enabled=no>
Note 1 : This option is only effective in CM mode and is ignored in VC mode.
Note 2 : In ASCII  : Variable length hex value just after the data field with an '@' prefix.
Note 3 : In BINARY : Appended as a two-byte big-endian value to a normal message.
Note 4 : Time is measured in millisecond resolution.
Note 5 : A timestamp is only appended (if enabled) to received CAN messages.
```

2.12 CAN Timeout Commands

It is possible, under heavy CAN bus loads or faulty CAN bus, to place the unit in a state where no more input parsing from the COM port can occur due to the internal buffers being full and the CAN TX not being able to proceed. Under this condition, a config command cannot be processed. This is to ensure that no CAN messages are lost while waiting for the CAN bus to become available.

In order to prevent this, a timeout parameter can be specified that will ensure additional input parsing can occur at the expense of some CAN messages. This will allow a CONFIG command sequence to be recognized even when the CAN bus is overloaded or faulty.

The command to set/get the timeout follows:

get can timeout: Gets the timeout setting.

set can timeout can_tx_busy=off: Disables timeout, will wait indefinitely (default)

set can timeout can_tx_busy=1000: Sets the timeout (10 to 10000 ms)

2.12.1 Get can timeout

```
Command : get can timeout
Desc : Gets the current CAN timeout settings.
Ex 1 : get can timeout
Resp : <A:can_tx_busy=off>
Note: 1) Timeout can range from 10 to 10000 ms.  Use 'off' to disable.
```

2.12.2 Set can timeout can_tx_busy=timeout

```
Command : set can timeout can_tx_busy=timeout
Desc : Sets the current CAN timeout parameter 'can_tx_busy'.
Ex 1 : set can timeout can_tx_busy=off
Resp : <A:can_tx_busy=off>
Ex 2 : set can timeout can_tx_busy=1000
Resp : <A:can_tx_busy=1000>
Note: 1) Timeout can range from 10 to 10000 ms.  Use 'off' to disable.
```

2.13 Get Sernum in Command Mode

You can enable and configure the CAN port side to respond to CAN requests for the serial number.

2.13.1 get sernumcmd cm

Gets the current 128-bit serial number in command mode.

```
Command : get sernumcmd cm
Desc : Gets the current CM-Mode 128-bit serial number command settings.
Syntax of arguments:
  state      : YES | NO
  ID         : STD | EXT identifier
  ID-RTR     : STD | EXT RTR | DATA identifier
  identifier : Hexadecimal value of CAN identifier
Ex 1 : get sernumcmd cm
Resp : <A:enabled=yes RX=EXT 1FFFFFFF RTR TX[0]=EXT 1FFFFFFF TX[1]=EXT 1FFFFFFF>
Ex 2 : get sernumcmd cm
Resp : <A:enabled=no RX=STD 1FF DATA TX[0]=STD 1FF TX[1]=STD 1FF>
```

2.13.2 set sernumcmd cm enable

Enables or disables the command mode serial number command.

There are three identifiers that can be specified:

The RX identifier to listen for (either DATA or RTR

The TX identifier for the first 64-bits of the 128-bit serial number

The TX identifier for the second 64-bits of the 128-bits serial number

This feature can be enabled/disabled/defined from within CONFIG mode.

If enabled, it is only active when in CM mode. In VC mode, it is automatically disabled.

Operation when enabled:

Received CAN messages are compared against the RX identifier specified during configuration. If there is a match (ID, type, RTR, etc.), then the unit will respond with two CAN messages using the two TX identifiers defined during configuration.

The identifiers can be all the same or different, so long as they respect the rules of the CAN bus protocol. For example, do not attempt so send two CAN identifiers with the same ID at the same time (unless at least one of them is RTR).

This allows the end-user to query the CAN device in a system and determine its presence.

Example #1

Three units are configured to have the same RX identifier in RTR mode. Each units as unique TX identifiers. When a node on the network issues a RX identifier request for serial numbers, the three units will all transmit their respective serial number messages.

```
Command : set sernumcmd cm enabled=state RX=ID | ID RTR TX[0]=ID TX[1]=ID
Desc : Enables or disables and define CM-Mode 128-bit serial number command.
Syntax of arguments:
  state      : YES | NO
  ID         : STD | EXT identifier
  ID-RTR     : STD | EXT RTR | DATA identifier
  identifier : Hexadecimal value of CAN identifier
Ex 1 : set sernumcmd cm enabled=yes rx=std 1FF rtr tx[0]=std 1FF tx[1]=std 1FFF
Resp : <A:enabled=yes RX=STD 1FF RTR TX[0]=STD 1FF TX[1]=STD 1FF>
Ex 2 : set sernumcmd cm enabled=yes rx=std 1FF data tx[0]=ext 1FFFFFFF tx[1]=ext 1FFFFFFF
Resp : <A:enabled=yes RX=STD 1FF DATA TX[0]=EXT 1FFFFFFF TX[1]=EXT 1FFFFFFF>
Note 1 : Arguments are not case sensitive.
Note 2 : One or more arguments can be passed in one command.
Note 3 : Response always shows all arguments regardless of how many were passed.
```

2.14 Profiles

Users can now assign short text notes to each profile to aid in remembering the settings.

The command to set/get profile notes follows:

profile note: Returns the note for the currently active profile

profile note #: Returns the note for the specified profile number (0 to 9)

profile note all: Returns a list of all profile notes.

Note : When 'all' is specified, the return format depends on whether the command was entered in EDIT or ANGLED mode. If in edit mode, it is returned as a list with each profile note on a separate line. In angled mode, it is returned as a single line between angle brackets.

profile note="Text": Sets the profile note for the current profile to "Text" (default "")

profile note #="Text": Sets the specified profile note to "Text" (0 to 9)

2.14.1 Set Active Profile Number

```
Command : profile #
Desc : Sets the active profile.
There are ten profiles numbered from 0 to 9.  This command determines which profile
will be active in CM or VC operating mode.
Ex 1 : profile 1
Resp : <A:Active profile 1>
```

2.14.2 Copy Profile

```
Command : profile copy # #
Desc : Copies source profile to dest profile.
The first number is the source profile and the second number is the destination profile.
Ex 1 : profile copy 1 2
Resp : <A:Profile 1 copied to 2>
```

2.14.3 Profile Note

profile note: Returns the note for the currently active profile

```
Command : profile note
Desc : Returns note for current profile.
Ex 1 : profile note
Resp : <A:0="text">
```

2.14.4 Profile Note

profile note #: Returns the note for the specified profile number (0 to 9)

```
Command : profile note #
Desc : Returns note for specified profile number.
Ex 1 : profile note 1
Resp : <A:1="text">
```

2.14.5 Profile Note All

profile note all: Returns a list of all profile notes.

```
Command : profile note all
Desc : Returns note for all profiles.
Ex 1 : profile note all
Resp :
<A:
0="text"
1="text"
2="text"
3="text"
4="text"
5="text"
6="text"
7="text"
8="text"
9="text"
>
Ex 2 : <profile note all>
Resp : <0="text" 1="text" 2="text" 3="text" 4="text" 5="text" 6="text" 7="text" 8="text" 9="text">
Note 1 : Angled-mode response is a single-line.  Edit-mode response places each note on a separate line.
```

2.14.6 Profile Note Text

profile note="Text": Sets the profile note for the current profile to "Text" (default "")

```
Command : profile note="text"
Desc : Assigns "text" to current profile note.
Ex 1 : profile note="text"
Resp : <A:0="text">
Note 1 : To set a blank note, assign "".
Note 2 : Note must be ASCII printable and cannot contain '<' , '>' , or '"' characters.
```

2.14.7 Profile Note #=Text

profile note #="Text": Sets the specified profile note to "Text" (0 to 9)

```
Command : profile note #="text"
Desc : Assigns "text" to specified profile note.
Ex 1 : profile note 0="text"
Resp : <A:0="text">
Note 1 : To set a blank note, assign "".
Note 2 : Note must be ASCII printable and cannot contain '<' , '>' , or '"' characters.
```

2.14.8 Set Active Profile to Default

```
Command : default
Desc : Sets the active profile to default settings.
Ex 1 : default
Resp : <A:Current profile (1) set to defaults>
```

2.14.9 Set Specific Profile to Default

```
Command : default #
Desc : Sets the spcified profile to default settings.
Ex 1 : default 2
Resp : <A:Profile (2) set to defaults>
```

2.14.10 Set All Profiles to Default

```
Command : default all
Desc : Sets all profiles to default settings.
Ex 1 : default all
Resp : <A:All profiles set to defaults>
```

2.15 CAN Sernum in Config Mode

2.15.1 CAN SERNUM

The SERNUM command provides a means for the user to query the CAN WIFI for its unique 128-bit serial number. The GET SERNUM command only works while in CONFIG mode and always returns the serial number over the COM port.

2.15.2 Get Serial Number

```
Command : get sernum
Desc : Returns a 128-bit hex device serial number.
Ex 1 : get sernum
Resp : <A:16 16 02 1E 53 56 12 F0 4E BB 35 4D F5 00 00 00>
```

2.15.3 Get Version Number

```
Command : get version
Desc : Returns hardware, bootloader, and application version information.
Ex 1 : get version
Resp : HW=A BOOT=1.00 APP=1.00>
```

CAN WiFi will show HW=A BOOT=1.01B APP=1.01B

2.16 Test Options

2.16.1 Cycling LED Indicators

To confirm that all LED indicators are working, this diagnostic allows the CAN-TX/RX and COM-TX/RX LED indicators to cycle continuously.

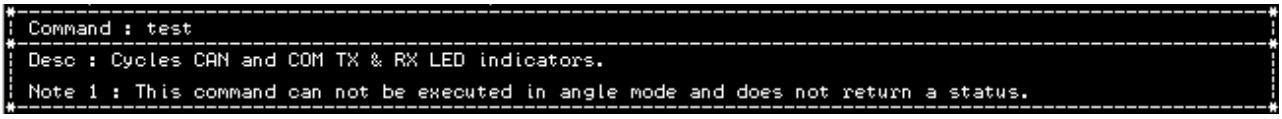
To terminate the diagnostic, press any key or the config button.

2.16.2 Generating CAN Square Wave

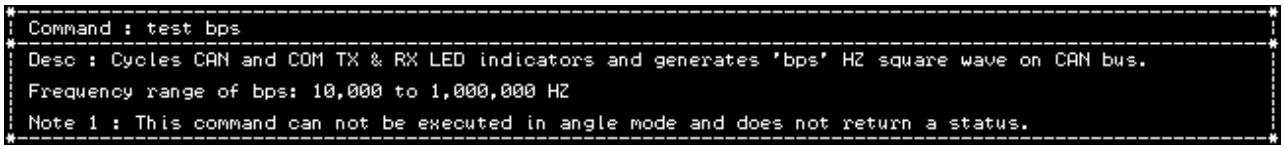
The CAN WIFI can generate a square wave output onto the CAN bus arbitrary frequencies. It is useful for measuring propagation delay through cabling with the aid of an oscilloscope and can be used to evaluate bus termination.

The square wave frequency can be in the range of 10,000 to 1,000,000 Hz.

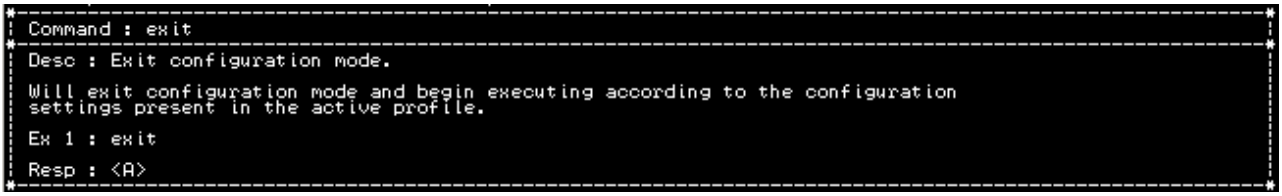
2.16.3 Test LEDs



2.16.4 Test LEDs and CAN Bus



2.16.5 Exit



2.17 Technical Support

If you are experiencing a problem, please read the user manual and other technical document supplied on the product CD. If you are unable to solve the problem, please contact technical support.

2.17.1 Live Chat

Go to the Grid Connect web page at gridconnect.com. Go to the bottom of the page and locate the **Support** link button. From there you can connect to a live chat session. After hours and when a representative is not available, clicking on Live Chat will allow you to send a message which we will answer as soon as we can. See the table below for hours of operation.

2.17.2 Contact Information

We are located in a far western suburb of Chicago and are on Central Standard Time. Our location allows us to best service all of North America.

Grid Connect, Inc.
1630 W. Diehl Road
Naperville, Illinois 60563 USA

+1 (800) 975-GRID (4743) USA Toll Free
+1 (630) 245-1445 Phone
+1 (630) 245-1717 Fax

Hours of Operation

Monday - Friday
Business Hours: 7:30 a.m. - 5:00 p.m. CST
Tech Support Hours: 8 a.m. - 5:30 p.m. CST