

About

The CAN source files (CAN2.c & CAN2.h) and this document (CAN.doc / CAN.pdf) are copyright John Harding (Copyright 2012). These files are intended for use with MMBasic or one of its derivatives (such as DMBasic). Geoff Graham is the overall author of MMBasic.

The information herein is provided in accordance with License X. Please see "License.txt" which should have been distributed with the source code. If using a compiled version then the license of the compiled version supersedes the original source license. (Note, if you alter and or compile the CAN2.c / CAN2.h files you must comply with the licensing as-is or seek a different licensing agreement from the copyright holder).

The "normal distribution method" is as a compiled HEX file intended for a DuinoMite build of MMBasic 4.x or later. The normal distribution is licensed following Geoff Graham's license requirements for his MMBasic (in brief: the hex can be used for any purpose, whereas source code is available for personal use by using the form at: <http://mmbasic.com/source.html>)

These commands are designed as a replacement to the original CAN commands that were provided with DMBasic. These new commands provide finer grained control of the CAN interface (most noticeably the ability to filter channels and the dynamic management of memory). These new commands also break the "licensing deadlock" created by incompatible licensing of the two versions of the BASIC interpreter (DMBasic by Olimex and MMBasic by Geoff Graham). As such they can be included in the official MMBasic distribution by Geoff Graham.

For backward compatibility it is possible to implement the original CAN commands as BASIC subroutines in MMBasic 4.0. It should also be straightforward to port these commands into DMBasic – however, to exist alongside the original commands some care will need to be taken to change the memory usage. Also, the latest (at time of writing) version of DMBasic is 2.7 and does not include support for BASIC subroutines.

These commands are a complete re-write of the CAN interface. Any similarity between these source files and the originals is because:

- (a) Both versions fit into the same MMBasic defined infrastructure.
- (b) Both versions use Microchips Peripheral Library (plib).
- (c) Both versions are attempting to solve the same problem (provide a high level interface for accessing the DuinoMite Mega CAN hardware).

However, without the DuinoMite Mega by Olimex and Frank Voorburg's original CAN commands this implementation would never have been written. In turn, without Geoff Graham's MaxiMite the DuinoMite wouldn't have existed. Thanks guys!

Peace, love and plug-ins,
John Harding, September 2012.

<http://prianschat.com/threads/yapip-recreating-peefs-approach.109724/>

CAN Commands Overview

These commands provide access to the CAN hardware available on the DuinoMite Mega. The DuinoMite Mega exposes one of the two CAN modules on the PIC microprocessor. A CAN module supports up to 32 channels, each channel can be configured to transmit or receive data. For receiving data the CAN module writes received messages into a different FIFO buffer for each channel. One entry in a FIFO buffer is 16 bytes long (4 address bytes, 4 length bytes, 8 data bytes). These commands allow the user to specify the number of records in each FIFO buffer and the internals keep track of the memory requirements and automatically allocates memory from the available heap. A command is provided to free the allocated memory when the CAN interface is no longer required.

Note that CAN channels are numbered from 0 to 31 and that, per the Microchip documentation, CAN channels should be configured contiguously from 0 upwards. I.e. if 5 channels are required use channels 0, 1, 2, 3 & 4; and not 1, 2, 3, 4 & 5 or 3, 5, 10, 22, 31 or any other combination.

The CAN module is first configured by placing it in configuration mode and then issuing configuration commands. When the desired configuration is set up then the CAN module is enabled. Once enabled the RX and TX commands can be used to receive and send data on the bus. Once the CAN module is finished it should be disabled and the memory associated with the FIFO buffers should be freed (for example at the end of the application)

An RX channel can either have a filter for a specific id or be configured to receive all messages. In the latter case the user can request the id of the received command and then perform processing based on the id of the command. However, on a CAN bus with moderate to high utilization, more reliable operation will be achieved by setting up multiple channels with each channel filtering for one id of interest. This is because of the relatively small buffer sizes and the relatively slow operation of the BASIC interpreter – when trying to receive all messages into one buffer the buffer overruns and data is lost. Furthermore, when a channel is being filtered for a single ID it may be appropriate to set the FIFO buffer size to just 1 record, especially if the goal is to update a single value display. This will always give you the most up-to-date data.

The examples provided were designed for (and tested on) a second generation Toyota Prius (Model years 2004 – 2009). For a feature-rich application please see: <http://priuschat.com/threads/my-duinomite-mega-canview-v4-equivalent-project.112429/> which provides details of a complete application developed by John Lopez.

Throughout this overview I have referred to “CAN commands” (plural). In fact there is only one true CAN command – the other commands are “sub-commands” from this one master. By structuring the commands this way we have less impact on the limitation for the number of top-level/master commands that MMBasic can support.

The commands are as follows:

Informational

```
CAN  
CAN PRINTCONFIG
```

Setup

```
CAN CONFIG ok  
CAN SETSPEED speed, ok  
CAN ADDRCHNL channel_num, can_id, msg_type, buffer_size, ok  
CAN ADDTXCHNL channel_num, buffer_size, ok  
CAN ENABLE ok
```

Read / Write

```
CAN RX channel_num, can_id, msg_type, length, data(), ok  
CAN RX channel_num, data(8), ok  
CAN TX channel_num, can_id, msg_type, length, data(), ok
```

Teardown

```
CAN DISABLE ok  
CAN FREE
```

Note: Examples are provided after the command documentation rather than including snippets for each individual command. This allows the reader to see all the commands in context.

Command: CAN

Category: Informational

Arguments: None

Description: Displays a list of the available commands.

Command: CAN PRINTCONFIG

Category: Informational

Arguments: None

Description: Displays details of the current configuration. The details displayed are module status (online or offline), speed and configured channels.

Command: CAN CONFIG ok

Category: Setup

Arguments:

ok (output) – 1 if successful, 0 otherwise

Description: Clears any pre-existing configuration and puts the CAN module into configuration mode.

Command: CAN SETSPEED speed, ok

Category: Setup

Arguments:

speed (input) – baud rate in kbps from 10,000 to 1,000,000

ok (output) – 1 if successful, 0 otherwise

Description: Sets the baud rate of the CAN connection. Value provided is bits per second with a minimum of 10kbps and a maximum of 1Mbps.

Command: CAN ADDRCHNL channel_num, can_id, msg_type, buffer_size, ok

Category: Setup

Arguments:

channel_num (input) – a CAN channel from 0 to 31

can_id (input) – a CAN id to filter for (set to 0 to receive all CAN messages)

msg_type(input) – 0 for standard 11-bit IDs, 1 for extended 29-bit IDs

buffer_size(input) – size of the FIFO buffer for this channel expressed as number of records

ok (output) – 1 if successful, 0 otherwise

Description: Configures the specified channel as a receive channel. To filter for an individual id provide the CAN id of interest, to receive all messages pass in a zero id. If you're monitoring a single id (with this channel) and want to act on the latest data set the buffer size to 1. Larger buffer sizes can be set to capture more data – note that no indication is given when buffer overrun occurs (the oldest data is simply discarded).

Command: CAN ADDTXCHNL channel_num, buffer_size, ok

Category: Setup

Arguments:

channel_num (input) – a CAN channel from 0 to 31

buffer_size(input) – size of the FIFO buffer for this channel expressed as number of records

ok (output) – 1 if successful, 0 otherwise

Description: Configures the specified channel as a transmit channel. Normally a buffer size of 1 is sufficient. However, larger sizes allow you to separate the construction and buffering of transmissions from the actual transmission.

Command: CAN ENABLE ok

Category: Setup

Arguments:

ok (output) – 1 if successful, 0 otherwise

Description: Once the configuration is complete call this command to put the CAN module into normal operating mode and ready to receive or transmit data.

Command: CAN RX channel_num, data(8), ok

Category: Read/Write

Arguments:

channel_num (input) – a CAN channel from 0 to 31 for a channel previously configured for RX

data(8) (output) – an array to receive the data from the FIFO record

ok (output) – 1 if successful, 0 if no data available or other failure occurs

Description: This command is intended to read the data only from a channel that has been previously configured to monitor for a given ID (hence the id is already known and doesn't need to be retrieved from the buffer).

Command: CAN RX channel_num, can_id, msg_type, length, data(), ok

Category: Read/Write

Arguments:

channel_num (input) – a CAN channel from 0 to 31 for a channel previously configured for RX

can_id (output) – the CAN id of the message read from the FIFO buffer

msg_type (output) – the message type of the message read from the FIFO buffer

length (output) – the amount of data read (between 0 and 8 bytes)

data(8) (output) – an array to receive the data from the FIFO record

ok (output) – 1 if successful, 0 if no data available or other failure occurs

Description: This command is intended to read information from the FIFO buffer of a channel that has been previously configured to receive all messages – hence the need to provide variables to retrieve the full information about the message.

Command: CAN TX channel_num, can_id, msg_type, length, data(), ok

Category: Read/Write

Arguments:

channel_num (input) – a CAN channel from 0 to 31 for a channel previously configured for RX

can_id (input) – the CAN id to send

msg_type (input) – the message type being sent (0=SID, 1=EID)

length (input) – the amount of data being sent (between 0 and 8 bytes)

data(8) (input) – an array of data bytes to send (values between 0 and 255)

ok (output) – 1 if successful, 0 otherwise

Description: Places data into the FIFO buffer for this channel. Data will be sent on the bus when the CAN module detects the bus is available (i.e. not busy).

Command: CAN DISABLE ok

Category: Teardown

Arguments:

ok (output) – 1 if successful, 0 otherwise

Description: Puts the module into offline mode, but does not destroy the configuration. This can be used to stop all processing of CAN messages while another processor intensive task takes place. CAN ENABLE can then be called to re-enable the pre-existing configuration.

Command: CAN FREE

Category: Teardown

Arguments: none

Description: Takes the module off line and frees all memory associated with the existing configuration.

EXAMPLE ONE – Minimal example of reading one channel

```
' (c) John Harding, 2012 - see license.txt for
' licensing details

' Example designed for Gen 2 Prius

' Configures connection speed to 500kbps and
' monitors for CAN id 52Ch when data is received
' we calculate the ECT from the appropriate data
' bytes.

' Note that the period of this message is
' approximately 1Hz
```

```
Cls
Dim ok
Dim data(8)
Dim ect

CAN CONFIG ok
CAN SETSPEED 500000, ok
CAN ADDRCHNL 0, &h52C, 0, 1, ok
CAN ENABLE ok

Timer = 0
Do
  If (Inkey$ = "q") Then Exit
  CAN RX 0, data(0), ok
  If (ok=1) Then
    Print Timer ": ECT = " (data(1)/2)
  EndIf
Loop

CAN FREE

End
```

EXAMPLE TWO – Example of reading all channels and displaying just one (but don't do this!)

```
' (c) John Harding, 2012 - see license.txt for
' licensing details

' Example designed for Gen 2 Prius

' Configures connection speed to 500kbps and
' configures a channel to receive all messages
' into a FIFO buffer with 32 records.
'
' When a message with id 52Ch is received
' we calculate the ECT from the appropriate data
' bytes.

' Note that the period of this message is
' approximately 1Hz but that we receive many
' wrong ids before we get the message we want

' This example is provided to contrast with
' example 1. It is suggested to use example 1
' as the basis for your code.

Dim ok
Dim data(8)
Dim id
Dim typ
Dim length

CAN CONFIG ok
CAN SETSPEED 500000, ok
CAN ADDRCHNL 0, 0, 0, 32, ok
CAN ENABLE ok
Timer=0
Do
  q$ = Inkey$
  If (q$="q") Then Exit
  CAN RX 0, id, typ, length, data(0), ok
  If (ok=1) Then
    If (id=&H52C) Then
      Print " "
      Print Timer ": " Hex$(id) " : " length " : ECT= " data(1) / 2 "
C"
    Else
      Print Timer ": " Hex$(id) " ";
    Endif
  EndIf
Loop
CAN FREE
End
```

EXAMPLE THREE – More complex example of reading several channels

Watch this space!

EXAMPLE FOUR – Subroutine for requesting PIDs

Watch this space!

EXAMPLE FIVE – Backward compatibility subroutines

Watch this space!