Megasquirt 29bit CAN protocol

Dated: 2015-01-20

This version of the documentation applies to:

- MS2, Microsquirt, Microsquirt-module, MSPNP2
  running MS2/Extra firmware 3.3.x
  OR
- MS3, MS3-Pro, MS3-Gold, MSPNP-Pro
  running MS3 firmware 1.3.x
  OR
- Microsquirt transmission control code version 0.025 or later

The details may also apply to other firmware versions, but differences may exist.
MS2/BG and derived firmwares support a subset of the protocol described here.

Current Megasquirt firmwares (MS2/Extra 3.4 and MS3 1.4) also support 11bit broadcasting. Dashes or dataloggers that need to receive data only are strongly encouraged to use that instead of this 29bit protocol. The 11bit broadcasting protocol spec can be downloaded from www.msextra.com/doc/pdf/

Table of Contents

1 Introduction

This document covers the 'on-the-wire' 29bit CAN protocol used between Megasquirt devices. This is a proprietary CAN protocol and has fundamental differences from the most commonly used automotive standards or protocols:

- 29 bit headers are used
- The message identifier bits are used for addressing.
- Messages are request/response - there is no broadcasting

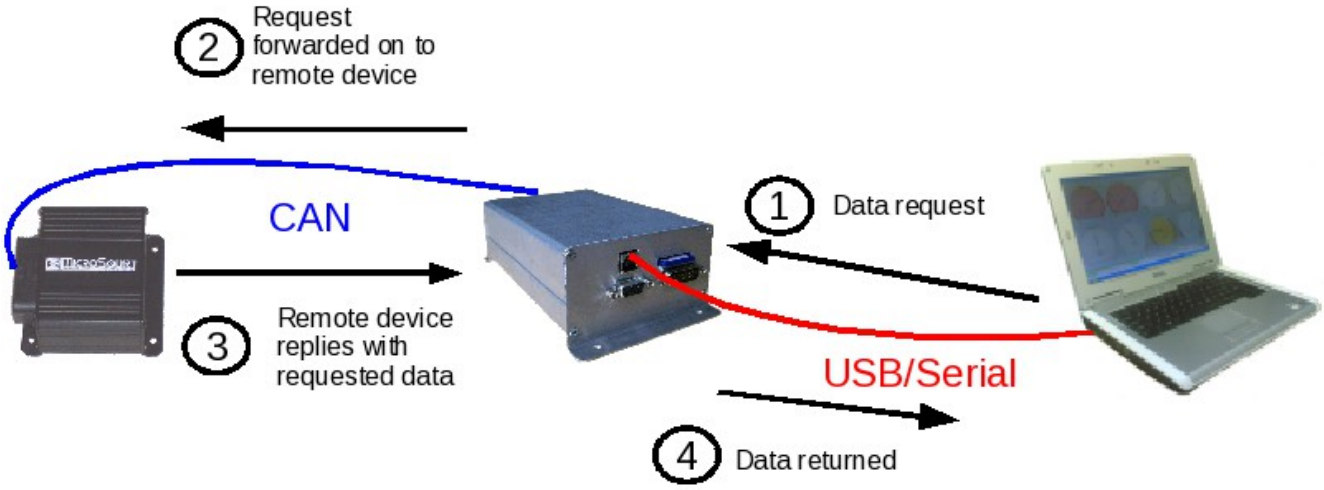
There are two main methods in which CAN communications are used within the Megasquirt system

1.1 Device to device

Devices on the CAN network request exchange messages to share data.

1.2 Passthrough

This is used to tune a remote device. The tuning computer is connected to one device on the CAN network and messages are relayed over CAN to the remote device.



2 Background concepts

The Megasquirt CAN system builds on many of the concepts (CANid, tables etc.) that are covered in the Megasquirt serial protocol manual.

Each CAN packet contains a maximum of 8 bytes (per the Bosch CAN protocol.)

2.1 Terms

2.1.1 CANid

The Megasquirt identifier of the device. The master ECU is always zero.

Other 'well known' ids

1 GPIO transmission controller

2 GPIO board

4 JBPerf TinyIOx

5 JBperf IO-x

7 Microsquirt transmission controller

2.1.2 Table

Within the Megasquirt memory map various regions are referred to as tables. This will vary depending on firmware revision and features. Consult the "ini" file supplied with the firmware as the final authority.

MS3 table list as per firmware 1.3.x

Table no.	Size	Internal name	Function
0	2048	cltfactor	Calibration table for CLT sensor.
1	2048	matfactor	Calibration table for MAT sensor.
2	1024	egofactor	Calibration table for AFR/EGO sensor.
3	2048	maffactor	Calibration table for MAF sensor.
4	1024	flash4	Tuning data. (TunerStudio 'page 1')
5	1024	flash5	Tuning data. (TunerStudio 'page 2')
6	-	canbuf	Used for CAN passthrough mainly.
7	varies	outpc / datax	Realtime data and data exchange.
8	1024	flash8	Tuning data. (TunerStudio 'page 3')
9	1024	flash9	Tuning data. (TunerStudio 'page 4')
10	1024	flash10	Tuning data. (TunerStudio 'page 5')
11	1024	flash11	Tuning data. (TunerStudio 'page 6')
12	1024	flash12	Tuning data. (TunerStudio 'page 7')
13	1024	flash13	Tuning data. (TunerStudio 'page 8')
14	60	Signature	Version and copyright string.
15	20	RevNum	Serial format string.
16	-	buf2	Special use.
17	1024	-	SDcard control.

18	1024	flash18	Tuning data. (TunerStudio 'page 9')
19	1024	flash19	Tuning data. (TunerStudio 'page 10')
20	2056	-	SDcard file readback.
21	1024	flash21	Tuning data. (TunerStudio 'page 11')
22	1024	flash22	Tuning data. (TunerStudio 'page 12')
23	1024	flash23	Tuning data. (TunerStudio 'page 13')
24	1024	flash24	Tuning data. (TunerStudio 'page 14')
25	1024	flash25	Tuning data. (TunerStudio 'page 15')
26	1024	trimpage	Read only data. (TunerStudio 'page 16')
27	1024	flash27	Tuning data. (TunerStudio 'page 17')
28	1024	flash28	Tuning data. (TunerStudio 'page 18')-
29	-	-	-
30	-	-	-
31	-	-	-
0xf0	1024	-	Tooth logger data.
0xf1	1024	-	Trigger logger data.
0xf2	1024	-	Composite logger data.
0xf3	1024	-	Sync error composite logger data.
0xf4	1024	-	MAP logger data.
0xf5	1024	-	MAF logger data.
0xf6	1024	-	Engine logger data.
0xf7	1024	-	Engine logger + MAP data.
0xf8	1024	-	Engine logger + MAF data.

Note that early Megasquirt firmwares only supported up to table 15. This means that you cannot use Megasquirt-3 as a 'slave' device with one of the older firmwares (MS2/BG, MShift) as the pass-through master.

Megasquirt-3 holds all tuning data pages in RAM concurrently. Data may be written to arbitrary tables and offsets. It will be made permanent (to that table) by a MSG_BURN command.

Calibration tables are held in flash only and the data within the table needs to be written sequentially to allow the firmware to erase and write the data correctly.

Megasquirt-2 (MS2/Extra) holds a single tuning page in RAM at any one time. Special care must be used by tuning software to allow for this. When a read or write command is issued to a different table, the firmware reads that data from flash to RAM. This will lose any previous changes to a RAM copy of a table if it has not been written to flash.

2.1.3 Offset

This is the address offset within a table, starting at 0. 16 bits big-endian.

2.1.4 Size

The number of bytes to read or write. 16 bits big-endian. Starting at 1 up to the maximum table size.

On Megasquirt-2 the maximum is 128 bytes, so to read a 1024 byte page a sequence of commands will be

required. e.g.

read 128 bytes at offset 0

read 128 bytes at offset 128

read 128 bytes at offset 256 etc.

2.1.5 Serial version

The version number of this protocol. Determine with MSG_PROT command. (Presently 2.)

2.1.6 Table blocking factor

The maximum size used to write to tables. Determine with MSG_PROT command. (At time of writing, MS2 = 256, MS3 = 2048)

2.1.7 Write blocking factor

The maximum size used for general tuning data reads and writes. Determine with MSG_PROT command. (At time of writing, MS2 = 256, MS3 = 2048)

3 Message formats

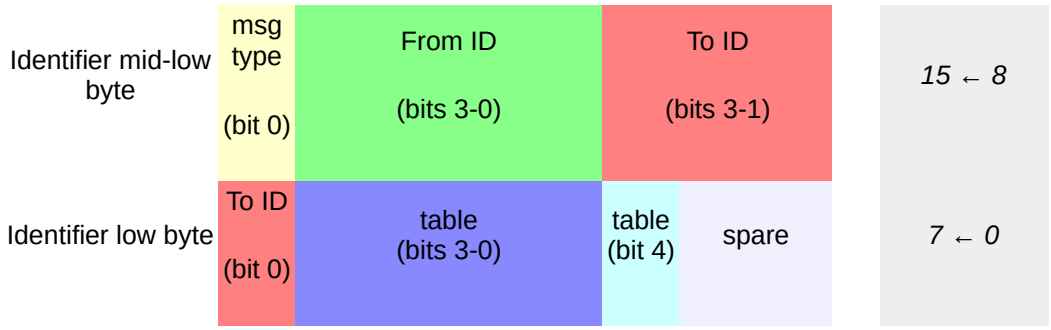
3.1 Core message

As received inside the chip (Freescale S12):

Register Bits →	7	6	5	4	3	2	1	0	Header Bits
Identifier byte 0	offset (bits 10-3)								*28 ← 21*
Identifier byte 1	offset (bits 2-0)			SRR (1)	IDE (1)	message type			*20 ← 15*
Identifier byte 2	From ID				To ID				*14 ← 7*
Identifier byte 3	table (bits 3-0)				table (bit 4)	spare		RTR	*6 ← 0, RTR*

Packed into a standard 29bit identifier:

Register Bits →	7	6	5	4	3	2	1	0	Header Bits
Identifier high byte	unused			offset (bits 10-6)					*28 ← 24*
Identifier mid-high-byte	offset (bits 5-0)						message type		*23 ← 16*



Here, the usage of the message identifier to communicate addressing is shown.

offset: the data offset within the table to send to/from

message type:

Value	Name	Function
0	MSG_CMD	A 'poke' message to deposit data into memory.
1	MSG_REQ	A request message for data.
2	MSG_RSP	A reply to a request, effectively the same as MSG_CMD.
3	MSG_XSUB	Not implemented
4	MSG_BURN	A request to burn data to flash.
5	OUTMSG_REQ	A request for an outmsg set of data.
6	OUTMSG_RSP	A response of an outmsg set of data.
7	MSG_XTND	An extended message, message number is in first data byte.

message type: (extended)

In extended messages, the message number is held in the first data byte.

Value	Name	Function
8	MSG_FWD	Message to send data out of the serial port.
9	MSG_CRC	Request for the CRC of a data table.
11	-	-
12	MSG_REQX	A request message for data (tables 16-31)
14	MSG_BURNACK	Response message for burn.
0x80	MSG_PROT	Command for getting the protocol version number
0x81	MSG_WCR	Not implemented.
0x82	MSG_SPND	Command for suspending and resuming CAN polling to a device.

From id: The CANid of the sending device.

To id: The CANid of the destination device. (Devices can use a mask on the identifier to only accept messages intended for them.)

Table: The internal table number in which to store the data (write commands) or read data from (read commands.)

Data length: The data length bits in the CAN message are used as per the standard to hold the number of data bytes.

Data bytes:

Register Bits →	7	6	5	4	3	2	1	0
Data 0	Depends on message type							
Data 1								
Data 2								
Data 3								
Data 4								
Data 5								
Data 6								
Data 7								

3.2 Message types

Each message will now be explained in detail with the usage of the data bytes.

3.2.1 MSG_CMD

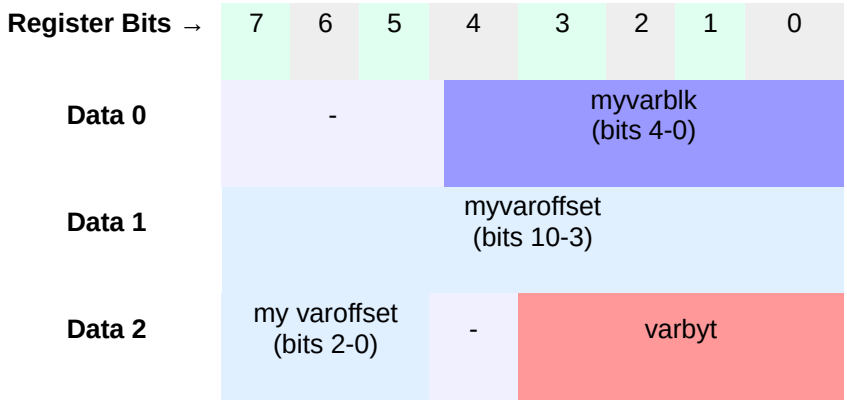
A 'poke' message to deposit data into memory.

The message identifier covers all of the addressing - this determines the table number and offset within the table where the data bytes will be stored. There is no checking, if the table and offset are valid, the data is written to memory.
Data length indicates the number of bytes.

3.2.2 MSG_REQ

A request message for data.

The message identifier covers all of the addressing. The block and offset refer to where to fetch the data from.
Data length is 3.



myvaroffset: This is the offset field in the MSG_RSP generated - i.e. the offset to store the reply message data at.
myvarblk: This is the table field in the MSG_RSP generated - i.e. the table to store the reply message data in.
varbyt: This is how many data bytes to send back in the MSG_RSP

3.2.3 MSG_RSP

A reply to a request, this is exactly the same message format as MSG_CMD.

3.2.4 MSG_XSUB

Placeholder message to execute some code on a remote device. Not implemented.

3.2.5 MSG_BURN

A request to burn tuning data to flash. Not used for calibration tables.

The message identifier covers all of the addressing.
Data length is 0.
The data region is not used.
The offset field is not used.

This instructs a remote device to burn flash table X. This is used when tuning a remote device. The tuning computer will already have used the serial 'r' and 'w' commands to read and modify data in that table. After the fresh data is written back, the serial 'b' command will be used which triggers a MSG_BURN command.
In firmwares after Jan 2015, a MSG_BURNACK message is sent back after the burn completes.

3.2.6 OUTMSG_REQ

A request for a consolidated group of data.
At this time, Megasquirt ECUs never sends this message type, but can respond to it. A single request message triggers one or many responses for more efficient bandwidth usage. However, the "outmsg" blocks of data need to be pre-configured in the ECU and must match what the remote end is expecting. There is no existing method to synchronise these.
(Using the 11bit broadcasting system may be a better alternative.)

Unusually, the addressing in the identifier is used to indicate the return block and offset.
i.e.
Table: This is used as the table to send data back to.
Offset: This is used as the starting offset to send data back to.

Data length is 1.

Register Bits →	7	6	5	4	3	2	1	0
Data 0	msg_no							

msg_no: The outmsg message number to reply with.
The main ECU has a number of message tables to hold different datasets.

3.2.7 OUTMSG_RSP

A response with a consolidated group of data, usually one of many responses.
The exact data to return is pre-configured in the ECU.

The destination table and base offset were obtained in the OUTMSG_REQ message. The destination offset is incremented on each reply packet. The replies will be 8 bytes long until the final packet which may be shorter.

Register Bits →	7	6	5	4	3	2	1	0
Data 0	1-8 data bytes							
...								

3.2.8 MSG_XTND

This message type indicates that the actual message type is held in the first byte of the data region. The following 'extended' commands reflect that.
The original MS2/BG CAN implementation does not support and extended messages.

3.2.9 MSG_FWD

Message to send data out of the serial port.

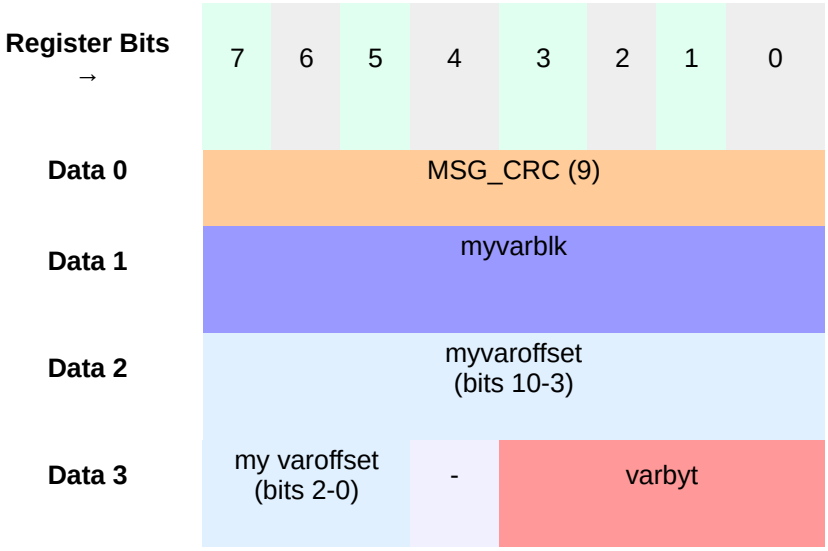
Register Bits →	7	6	5	4	3	2	1	0
Data 0	MSG_FWD (8)							
Data 1	1-7 data bytes							
...								

The data bytes are sent 'raw' without any error checked wrapper.

3.2.10 MSG_CRC

Request for the CRC of a data table. (Remote will reply with MSG_RSP.)
The message identifier covers all of the addressing.

Data length is 4.

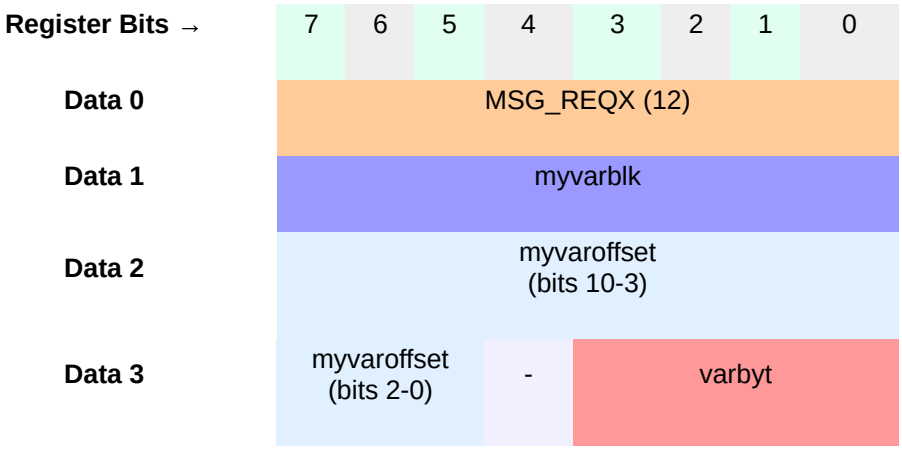


myvaroffset: This is the offset field in the MSG_RSP generated - i.e. the offset to store the reply message data at. (Currently always 0.)
myvarblk: This is the table field in the MSG_RSP generated - i.e. the table to store the reply message data in. (Currently always 6.)
varbyt: This is how many data bytes to send back in the MSG_RSP. (Currently always 4.)

3.2.11 MSG_REQX

A request message for data for tables > 31.
The Megasquirt firmware will automatically use this instead of MSG_REQ if a table > 31 such as the logger 'tables.' (Note that there is not presently a symmetrical MSG_CMDX.)

The message identifier covers all of the addressing.
Data length is 5.



Data 4	table

myvaroffset: This is the offset field in the MSG_RSP generated - i.e. the offset to store the reply message data at.
myvarblk: This is the table field in the MSG_RSP generated - i.e. the table to store the reply message data in.
varbyt: This is how many data bytes to send back in the MSG_RSP

3.2.12 MSG_BURNACK

Response message after a Burn action has fully completed. Only implemented in firmwares Jan 2015 or later.

Data length is 2.

Register Bits →	7	6	5	4	3	2	1	0
Data 0	MSG_BURNACK (14)							
Data 1								Value

Value: 1 = burn succeeded. 0 = burn failed.

3.2.13 MSG_PROT

Request for the protocol of the remote device. (Remote will reply with MSG_RSP.)
The message identifier covers all of the addressing.
The offset field is not used.
The table field is not used.

Data length is 4.

Register Bits →	7	6	5	4	3	2	1	0
Data 0	MSG_PROT (0x80)							
Data 1	myvarblk							
Data 2	myvaroffset (bits 10-3)							
Data 3	my varoffset (bits 2-0)			-	varbyt			

myvaroffset: This is the offset field in the MSG_RSP generated - i.e. the offset to store the reply message data at. (Currently always 0.)
myvarblk: This is the table field in the MSG_RSP generated - i.e. the table to store the reply message data in. (Currently always 6.)
varbyt: This is how many data bytes to send back in the MSG_RSP. This will be 1 for just the protocol version of 5 for protocol and block sizes.

Response when data length 1

Register Bits →	7	6	5	4	3	2	1	0
Data 0	Serial version (2)							

Response when data length 5

Register Bits →	7	6	5	4	3	2	1	0
Data 0	Serial version (2)							
Data 1	Table blocking factor (high byte)							
Data 2	Table blocking factor (low byte)							
Data 3	Write blocking factor (high byte)							
Data 4	Write blocking factor (low byte)							

3.2.14 MSG_SPND

Command for suspending and resuming CAN polling to a device.
Presently this is sent to CANid 15 as a broadcast address. That is not compatible with "CAN receiving."

Data length is 2.

Register Bits →	7	6	5	4	3	2	1	0
Data 0	MSG_SPND (0x82)							
Data 1								Value

Value: 1 = suspend CAN polling and broadcasting. 0 = resume.

4 Examples

4.1 Fetch serial signature (passthrough)

In this example, CANid 7 is fetching bytes from table 15 (serial signature) on CANid 0 as part of a serial/CAN-passthrough transfer. The received data is then passed on to the tuning computer via serial. In this case data is sent back to "offset 0". In the case of a device-to-device transfer, other offsets would be used in order to store the data locally.

Identifiers are shown as true on-the-wire identifiers.

Outbound request:

id=0xb878, DLC=3, Data = 0x06 0x00 0x08

From id = 7, To id = 0, Table = 0x0f, message type = 1 (MSG_REQ), offset = 0, my varblk = 6, my varoff =0, varbyt = 8

Remote replies with:

id=0x103b0, DLC=8, Data = 0x4d 0x53 0x33 0x20 0x46 0x6f 0x72 0x6f ("MS3 Form")

From id = 0, To id = 7, Table = 6, message type = 2 (MSG_RSP), offset = 0

Local device then stores this into table 6 at offset 0 (and then forwards to tuning computer by serial.)

Outbound request:

id=0x20b878, DLC=3, Data = 0x06 0x00 0x08

From id = 7, To id = 0, Table = 0x0f, message type = 1 (REQ), offset = 8, my varblk = 6, my varoff =0, varbyt = 8

Remote replies with:

id=0x103b0, DLC=8, Data = 0x61 0x74 0x20 0x30 0x34 0x33 0x35 0x2e ("at 0435.")

From id = 0, To id = 7, Table = 6, message type = 2 (MSG_RSP), offset = 0

Local device then stores this into table 6 at offset 0 (and then forwards to tuning computer by serial.)

Outbound request:

id=0x40b878, DLC=3, Data = 0x06 0x00 0x04

From id = 7, To id = 0, Table = 0x0f, message type = 1 (REQ), offset = 16, my varblk = 6, my varoff =0, varbyt =4

Remote replies with:

id=0x103b0, DLC=8, Data = 0x31 0x34 0x20 0x00 ("14 ")

From id = 0, To id = 7, Table = 6, message type = 2 (MSG_RSP), offset = 0

Local device then stores this into table 6 at offset 0 (and then forwards to tuning computer by serial.)

4.2 Fetch RPM (device-to-device)

In this example, CANid 7 is fetching RPM data from table 7 (realtime data) on CANid 0 to use internally.

The variable "RPM" happens to be available at CANid0 in table 7, offset 6.

The request stores that data into table 7, offset 2 on the CANid 7.

Identifiers are shown as true on-the-wire identifiers.

Outbound request:

id=0x18b838, DLC=3, Data = 0x07 0x00 0x42

From id = 7, To id = 0, Table = 7, message type = 1 (MSG_REQ), offset = 6, my varblk = 6, my varoff =2, varbyt = 2

Remote replies with:

id=0x903b8, DLC=2, Data = 0x14 0xfe (5374 RPM)

From id = 0, To id = 7, Table = 7, message type = 2 (MSG_RSP), offset = 2

Local device then stores this into table 7 at offset 2.

For a 3rd party device fetching data from Megasquirt devices, create a receive buffer array and assign it an arbitrary table number. e.g. 1. All outbound MSQ_REQ messages would then specify my_varblk = 1. The response message (MSG_RSP) generated by the Megasquirt will then be addressed back to your table 1. Listen for this and store the data into the given offset within the array. e.g.

```
if (var_blk == 1) {
    int x;
    for (x = 0 < ; x < DLC ; x++) {
        receive_buffer[offset + x] = CAN_DATA_BYTE[x];
    }
}
```

Range checking and other validation needs to be applied.