NGC CANbus Communication Specification

The NetGain Controls CANbus operates at 250Kbps. Refer to CAN2.0B for the data link layer, as well as SAE J1939. The complete NGC specification is described in this document.

Note: All numbers shown in this document are in decimal format unless otherwise specified with the prefixes "0x" for hexadecimal or "0b" for binary.

Physical Layer

The recommended connector is RJ-45. Twisted pair cable should be used. The pinouts for the connectors are shown in the following illustration and table.

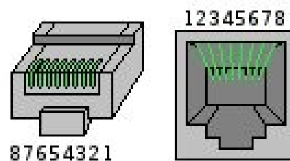

Illustration 1: RJ-45 Pinout Diagram

RJ-45 Pin #	Signal Name	Signal Description
1	CAN_H	Dominant High
2	CAN_L	Dominant Low
3	CAN_GND	Ground
4	Reserved	Reserved
5	Reserved	Reserved
6	CAN_SHLD	CAN Shield, optional
7	CAN_GND	Ground
8	CAN_V+	Power, optional

The WD1 RJ-45 connectors have routed to their pin 8 the WD1 constant 12V supply. Both these power pins are protected by a resettable fuse and can each provide up to 150mA.

CAN Identifier

CAN extended frame is used. The following table and descriptions detail the use of the extended frame.

Identifier – 11 bits											Identifier Extension – 18 bits																	
Priority			R	DP	PDU Format (PF)						PF		PDU Specific (PS)								Source Address (SA)							
3	2	1	1	1	8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1
28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

The above table does not show the SRR and IDE bits that separate the lower two bits of PF from the higher 6 bits. The SRR and IDE bits are defined by CAN2.0B, and are not affected by the J1939 specification. The major deviation in the NGC specification from J1939 is the lack of address claiming. Addresses are hard-coded into the devices. It is possible that future revisions will include provision for network start-up as per J1939.

Priority

There are 8 possible priorities. However, only three are used:

- 3 = High
- 5 = Medium
- 8 = Low

R: Reserved

Currently, only zero is used for this parameter.

DP: Data Page

Currently, only zero is used for this parameter.

PF: PDU Format

This parameter indicates whether the message is destination specific or broadcast. If the message is destination specific (PDU1), then PDU Format is 0-239. Otherwise, it is broadcast (PDU2), and falls in the ranges of 240-255. The majority of these values (both PDU1 and PDU2) are assigned under J1939 by the SAE. For destination specific, only 239 can be used for manufacturer-specific assignments. For broadcast, 255 is available for manufacturer-specific assignments. The bulk of the messages sent by NGC devices are broadcast.

PS: PDU Specific

The purpose of PDU specific depends on whether PF is PDU1 or PDU2.

- If PF is PDU1 (destination specific), then PS holds the destination address.
- If PF is PDU2 (broadcast), then PS holds the “Group Extension”.

SA: Source Address

The source address is the address of the device placing the message on the bus.

Network Addresses

The following devices are currently supported by the NetGain Controls CANbus protocol:

NAME	Source Address
Warp-Drive Speed Controller (WD1), unit “A”	88 (0x58)
Warp-Drive Speed Controller (WD1), unit “A”. Do not communicate to this address – for internal communications only.	89 (0x59)
Warp-Drive Speed Controller (WD1), unit “B”	90 (0x5A)
Warp-Drive Speed Controller (WD1), unit “B”. Do not communicate to this address – for internal communications only.	91 (0x5B)
Warp-Drive Speed Controller (WD2), unit “A”	92 (0x5C)
Warp-Drive Speed Controller (WD2), unit “A”. Do not communicate to this address – for internal communications only.	93 (0x5D)
Warp-Drive Speed Controller (WD2), unit “B”	94 (0x5E)
Warp-Drive Speed Controller (WD2), unit “B”. Do not communicate to this address – for internal communications only.	95 (0x5F)
Interface Module (IM1)	1 (0x01)
Speed Sensor Block (SSB)	28 (0x1C)
Elithion BMS (EBM)	50 (0x32)

Messages

The following table lists the possible messages currently used on the NGC CANbus network. Detailed descriptions of each message are listed after the table.

Message Designation	Message Description
0 (PDU1)	Clear error or warnings. Destination specific.
0 (PDU2)	IM time and date. Broadcast.
1 (PDU2)	Reserved.
2 (PDU2)	Reserved.
3.0 (PDU2)	Reserved.
3.1 (PDU2)	Reserved.
3.2 (PDU2)	Set WarP-Drive motor current limit.
3.3 (PDU2)	Set WarP-Drive battery current limit.
3.4 (PDU2)	Set WarP-Drive motor voltage limit.

Message Designation	Message Description
3.5 (PDU2)	Set WarP-Drive throttle unit.
3.6 (PDU2)	Reserved.
3.7 (PDU2)	Reserved.
3.8 (PDU2)	Reserved.
3.9 (PDU2)	Reserved.
3.10 (PDU2)	Reserved.
3.11 (PDU2)	Reserved
3.12 (PDU2)	Request non-interval data
3.13 (PDU2)	Set Warp-Drive motor current ramp.
4 (PDU2)	WarP-Drive power data.
5 (PDU2)	WarP-Drive sensor related data including brake and reverse.
6 (PDU2)	WarP-Drive supply voltages.
7 (PDU2)	WarP-Drive logged DTCs.
8 (PDU2)	Reserved
9 (PDU2)	WarP-Drive configured forward limits and model capabilities.
10 (PDU2)	WarP-Drive system info A.
11 (PDU2)	Warp-Drive system info. B.
12 (PDU2)	WarP-Drive configured reverse limits, motor current ramp, and installed throttle.
13 (PDU2)	Speed sensor information.
14 (PDU2)	Configuration byte for speed sensor device.

Destination specific message 0: WarP-Drive Error Clear. PS is usually the WarP-Drive address (88), and the SA is usually the IM address (1).

Message Identifier: 0xCEF58'SA'						
PF	**PS**	**SA**	**P**	**R**	**DP**	**Cycle Time (ms)**
239	destination	source	3	0	0	As Needed
Data Bytes: 1						
Position	**Data**		**Comments**			
Byte 0	0: Clear Errors					

Broadcast message 0: IM Time and Date.

Message Identifier: 0x20FF0001						
PF	**PS**	**SA**	**P**	**R**	**DP**	**Cycle Time (ms)**

255	0	1	8	0	0	200
Data Bytes: 7						
Position	**Data**			**Comments**		
Byte 0	Second: 0-59					
Byte 1	Minute: 0-59					
Byte 2	Hour: 0-23					
Byte 3	Day: 1-31					
Byte 4	Weekday: 0-6			Zero referenced to Monday.		
Byte 5	Month: 1-12					
Byte 6	Year: 0-127			Years from 1980.		

Broadcast message 1: Reserved.
Broadcast message 2: Reserved.
Broadcast message 3.0: Short Command 0 (Reserved)
Broadcast message 3.1: Short Command 1 (Reserved)

Broadcast message 3.2: Short Command 2. Used to set new motor current limit in controller. SA can be any device intended for setting new motor current limits, e.g. BMS.

Message Identifier: 0xCFF03'SA'						
PF	**PS**	**SA**	**P**	**R**	**DP**	**Cycle Time (ms)**
255	3	1, others	3	0	0	As Needed
Data Bytes: 5						
Position	**Data**			**Comments**		
Byte 0	2			Byte 0 indicates which short command		
Byte 1	Forward motor current limit, high byte					
Byte 2	Forward motor current limit, low byte					
Byte 3	Reverse motor current limit, high byte					
Byte 4	Reverse motor current limit, high byte					

Broadcast message 3.3: Short Command 3. Used to set new battery current limit in controller. SA can be any device intended for setting new battery current limit, e.g. BMS.

Message Identifier: 0xCFF03'SA'						
PF	**PS**	**SA**	**P**	**R**	**DP**	**Cycle Time (ms)**
255	3	1, others	3	0	0	As Needed
Data Bytes: 3						
Position	**Data**			**Comments**		
Byte 0	3			Byte 0 indicates which short command		
Byte 1	Battery current limit, high byte					

Byte 2	Battery current limit, low byte	

Broadcast message 3.4: Short Command 4. Used to set new motor voltage limit in controller. SA can be any device intended for setting new motor voltage limit.

Message Identifier: 0xCFF03'SA'						
PF	**PS**	**SA**	**P**	**R**	**DP**	**Cycle Time (ms)**
255	3	1, others	3	0	0	As Needed
Data Bytes: 5						
Position	**Data**			**Comments**		
Byte 0	4			Byte 0 indicates which short command		
Byte 1	Forward motor voltage limit, high byte					
Byte 2	Forward motor voltage limit, low byte					
Byte 3	Reverse motor voltage limit, high byte					
Byte 4	Reverse motor voltage limit, low byte					

Broadcast message 3.5: Short Command 5. Used to set desired throttle unit in controller. SA can be any device intended for setting new throttle unit. Consult factory before attempting to use throttle unit other than those listed in this section.

Message Identifier: 0xCFF03'SA'						
PF	**PS**	**SA**	**P**	**R**	**DP**	**Cycle Time (ms)**
255	3	1, others	3	0	0	As Needed
Data Bytes: 2						
Position	**Data**			**Comments**		
Byte 0	5			Byte 0 indicates which short command		
Byte 1	Desired throttle unit			0: NGC HEPA or HETA 1: Toyota Prius 2: NGC TPS-DP1 3: NGC WDT-CP (Cable-Pull) 4: Mazda MX5 5: Lokar Pedal		

Broadcast message 3.6: Short Command 6 (Reserved)
Broadcast message 3.7: Short Command 7 (Reserved)
Broadcast message 3.8: Short Command 8 (Reserved)
Broadcast message 3.9: Short Command 9 (Reserved)
Broadcast message 3.10: Short Command 10 (Reserved)

Broadcast message 3.11: Short Command 11. Used to set a the leakage options in the controller.

Message Identifier: 0xCFF03'SA'						
PF	**PS**	**SA**	**P**	**R**	**DP**	**Cycle Time (ms)**

255	3	1, others	3	0	0	As Needed
Data Bytes: 2						
Position	**Data**		**Comments**			
Byte 0	11					
Byte 1	Leakage option for leakage between pack and chassis		0: Take no action with detected leakage 1: Warn on detected leakage 2: Halt operation on detected leakage Leakage detection only available on industrial controller.			

Broadcast message 3.12: Short Command 12. Request Non-Interval WarP-Drive Data

Message Identifier: 0xCFF03'SA'						
PF	**PS**	**SA**	**P**	**R**	**DP**	**Cycle Time (ms)**
255	3	1, others	3	0	0	As Needed
Data Bytes: 1						
Position	**Data**		**Comments**			
Byte 0	12		As soon as the command is received by the WarP-Drive, it will send out data specified as "By Request"			

Broadcast message 3.13: Short Command 13. Used to set a new motor current ramp value in the controller.

Message Identifier: 0xCFF03'SA'						
PF	**PS**	**SA**	**P**	**R**	**DP**	**Cycle Time (ms)**
255	3	1, others	3	0	0	As Needed
Data Bytes: 2						
Position	**Data**		**Comments**			
Byte 0	13					
Byte 1	Motor current ramp, 1-250		Motor current ramp in amps/10msec.			

Broadcast message 4: WarP-Drive Power Data.

Message Identifier: 0x14FF04'SA'						
PF	**PS**	**SA**	**P**	**R**	**DP**	**Cycle Time (ms)**
255	4	88	5	0	0	100
Data Bytes: 8						
Position	**Data**		**Comments**			
Byte 0	Motor amps, high byte		Motor amps in amps.			
Byte 1	Motor amps, low byte					

Byte 2	Pack voltage, high byte	Pack voltage in volts.
Byte 3	Pack voltage, low byte	
Byte 4	PWM drive value, high byte	Maximum for PWM drive is 636.
Byte 5	PWM drive value, low byte	Maximum for PWM drive is 636.
Byte 6	Controller state	0: key off, 1: key on, 2: controller started
Byte 7	Last halt reason	Bits: 0: not started 1: halt errors 2: real time warnings 3: brake 4: 0 throttle amps 5: throttle not zeroed 6: reserved for future use 7: reserved for future use

Broadcast message 5: WarP-Drive Sensor Related Data.

Message Identifier: 0x14FF05'SA'						
PF	**PS**	**SA**	**P**	**R**	**DP**	**Cycle Time (ms)**
255	5	88	5	0	0	100
Data Bytes: 8						

Position	Data	Comments
Byte 0	Throttle channel 1, high byte	Throttle channel value is out of 1023. Voltage is (value / 1023) * 5.0.
Byte 1	Throttle channel 2, low byte	
Byte 2	Throttle channel 2, high byte	
Byte 3	Throttle channel 1, low byte	
Byte 4	Chillplate temperature	Sent in Celsius as signed 8-bit value.
Byte 5	Filmcap temperature	Sent in Celsius as signed 8-bit value.
Byte 6	Brake input	0: brake off 1: brake on
Byte 7	Reverse input	0: forward direction 1: reverse direction

Broadcast message 6: WarP-Drive Supply Voltages.

Message Identifier: 0x14FF06'SA'						
PF	**PS**	**SA**	**P**	**R**	**DP**	**Cycle Time (ms)**
255	6	88	5	0	0	100

Data Bytes: 5		
Position	**Data**	**Comments**
Byte 0	12V supply voltage	In volts x 10.
Byte 1	15V “A” supply voltage	In volts x 10.
Byte 2	15V “B” supply voltage	In volts x 10.
Byte 3	5V “A” supply voltage	In volts x 10.
Byte 4	5V “B” supply voltage	In volts x 10.

Broadcast message 7: WarP-Drive Logged Diagnostic Trouble Codes (DTC).

Message Identifier: 0x14FF07'SA'						
PF	**PS**	**SA**	**P**	**R**	**DP**	**Cycle Time (ms)**
255	7	88	5	0	0	100

Data Bytes: 8		
Position	**Data**	**Comments**
Byte 0	Byte 3 of Active DTCs	Each bit represents a trouble code. Up to 32 codes possible. A zero means the code is not set, a 1 means it is set. Codes offset by 1 (i.e. position 0 is DTC 1). See list of codes after this section.
Byte 1	Byte 2 of Active DTCs	
Byte 2	Byte 1 of Active DTCs	
Byte 3	Byte 0 of Active DTCs	
Byte 4	Byte 3 of Historical DTCs	
Byte 5	Byte 2 of Historical DTCs	
Byte 6	Byte 1 of Historical DTCs	
Byte 7	Byte 0 of Historical DTCs	

Diagnostic Trouble Codes (DTCs):
01: At least one throttle value out of allowable limits - pedal disconnected or failed.
02: One of the two throttle channels did not validate the off position.
03: Something wrong with the precharge wire connections - battery pack might be disconnected.
04: Precharge took longer than allowed time.
05: Input voltage too high.
06: Something not working with the pack voltage reading hardware.
07: 12V supply fell below acceptable minimum.
08: 15V “A” supply out of range (leakage fault on WD2)
09: 15V “B” supply out of range (sole 15V supply on WD2)
10: 5V “A” supply out of range
11: 5V “A” supply out of range
12: One of the two sensors aren't connected or temperature fell below -40C.
13 Current sensor appears to not be connected or working properly.
14: Reserved
15: Communications between onboard microcontrollers is not functioning correctly.
16: Main controller reported an overcurrent situation.
17: System was shut down while current was still flowing.
18: Current flow is substantially over what it's being commanded.
19: Gate drive not off when it should be.
20: Current reading is substantially higher than what it's commanded to be over a period of time

21: Desaturation was detected.
22: Firmware upgrade process failed.

Broadcast message 8: Reserved

Broadcast message 9: WarP-Drive Ratings and Forward Configured Limits.

Message Identifier: 0x14FF09'SA'						
PF	**PS**	**SA**	**P**	**R**	**DP**	**Cycle Time (ms)**
255	9	88	5	0	0	By Request
Data Bytes: 8						
Position	**Data**		**Comments**			
Byte 0	Current rating index		0x34: 1400A 0x32: 1200A 0x31: 1000A			
Byte 1	Voltage rating index		0x33: 360V 0x32: 260V 0x31: 160V			
Byte 2	Forward motor current limit, high byte					
Byte 3	Forward motor current limit, low byte					
Byte 4	Battery current limit, high byte					
Byte 5	Battery current limit, low byte					
Byte 6	Forward motor voltage limit, high byte					
Byte 7	Forward motor voltage limit, low byte					

Broadcast message 10: WarP-Drive System Info A.

Message Identifier: 0x14FF0A'SA'						
PF	**PS**	**SA**	**P**	**R**	**DP**	**Cycle Time (ms)**
255	10	88	5	0	0	By Request
Data Bytes: 8						
Position	**Data**		**Comments**			
Byte 0	Hardware version		Ranges from V0.00 to V25.5 (0 to 255).			
Byte 1	Firmware version, day		Tens position of day, in ASCII.			
Byte 2	Firmware version, day		Ones position of day, in ASCII.			
Byte 3	Firmware version, month		First letter of three-letter abbreviation, in ASCII.			
Byte 4	Firmware version, month		Second letter of three-letter abbreviation, in ASCII.			
Byte 5	Firmware version, month		Third letter of three-letter abbreviation, in ASCII.			

Byte 6	Firmware version, year	Tens position of year, in ASCII.
Byte 7	Firmware version, year	Ones position of year, in ASCII.

Broadcast message 11: WarP-Drive System Info B.

Message Identifier: 0x14FF0B'SA'						
PF	**PS**	**SA**	**P**	**R**	**DP**	**Cycle Time (ms)**
255	11	88	5	0	0	By Request
Data Bytes: 8						
Position	**Data**		**Comments**			
Byte 0	Public serial number, high byte					
Byte 1	Public serial number, low byte					
Byte 2	Exchange serial number, high byte					
Byte 3	Exchange serial number, low byte					
Byte 4	Capability upgrade count					
Byte 5	Firmware upgrade count					
Byte 6	Upgrade error count					
Byte 7	High temperature cycle count					

Broadcast message 12: WarP-Drive Reverse Configured Limits, Brake, Reverse, and Motor Current Ramp.

Message Identifier: 0x14FF0C'SA'						
PF	**PS**	**SA**	**P**	**R**	**DP**	**Cycle Time (ms)**
255	12	88	5	0	0	By Request
Data Bytes: 6						
Position	**Data**		**Comments**			
Byte 0	Reverse motor current limit, high byte					
Byte 1	Reverse motor current limit, low byte					
Byte 2	Reverse motor voltage limit, high byte					
Byte 3	Reverse motor voltage limit, low byte					
Byte 4	Motor current ramp		Motor current ramp in amps/10msec, 1-250.			
Byte 5	Installed throttle		0: NGC HEPA or HETA 1: Toyota Prius 2: NGC TPS-DP1 3: NGC WDT-CP (Cable-Pull) 4: Mazda MX5 5: Lokar Pedal			
Byte 6	Leakage option		0: Do nothing			

		1: Warn 2: Halt operation This byte only implemented on industrial controller.

Broadcast message 13: Transmitted by the speed sensing device.

Message Identifier: 0x14FF0D1C						
PF	**PS**	**SA**	**P**	**R**	**DP**	**Cycle Time (ms)**
255	13	28	5	0	0	100
Data Bytes: 8						
Position	**Data**			**Comments**		
Byte 0	Sensor 1, high byte			0.250 RPM/count, 0 offset.		
Byte 1	Sensor 1, low byte					
Byte 2	Sensor 2, high byte					
Byte 3	Sensor 2, low byte					
Byte 4	Sensor 1 divider			1 pulse/rev, 0 offset.		
Byte 5	Sensor 2 divider					
Byte 6	Software release minor					
Byte 7	Software release major					

Broadcast message 14: Used to configure the speed sensing device.

Message Identifier: 0x14FF0E'SA'						
PF	**PS**	**SA**	**P**	**R**	**DP**	**Cycle Time (ms)**
255	14	1, others	5	0	0	As Needed
Data Bytes: 4						
Position	**Data**			**Comments**		
Byte 0	CAN qualifier, high byte			Must be exactly 0x91.		
Byte 1	CAN qualifier, low byte			Must be exactly 0x19.		
Byte 2	Sensor 1 divider			0-255. A setting of 0 indicates the DIP switch setting will be used.		
Byte 3	Sensor 2 divider					

Example Code

The following shows some code snippets that might be useful in understanding how the protocol is applied. Please note, the code is not intended to be complete, but only to provide some assistance in

understanding how to implement J1939. Requests for additional pieces of code related to the following examples will not be fulfilled.

General protocol definitions and functions:

```
#define QUEUE_FULL 0
#define SUCCESS 1

#ifndef TX_BUFFER_SIZE
#define TX_BUFFER_SIZE 8 // adjust if the tx_buffer gets clogged - unlikely unless trying to run too much data through the pipe
#endif

#ifndef RX_BUFFER_SIZE
#define RX_BUFFER_SIZE 8 // adjust if messages are lost - unlikely if polled frequently enough
#endif

#define LOW_PRIORITY 8
#define MED_PRIORITY 5
#define HIGH_PRIORITY 3

#define NUM_BUFFER_BYTES 15
#define NUM_OVERHEAD_BYTES 7 // all bytes in the message prior to the data

#define MESSAGE_PRIORITY 0
#define MESSAGE_RESERVED 1
#define DATA_PAGE 2
#define PROTOCOL_FORMAT 3
#define PROTOCOL_SPECIFIC 4
#define MESSAGE_SOURCE_ADDRESS 5
#define MESSAGE_NUM_DATA_BYTES 6
#define MESSAGE_DATA0 7
#define MESSAGE_DATA1 8
#define MESSAGE_DATA2 9
#define MESSAGE_DATA3 10
#define MESSAGE_DATA4 11
#define MESSAGE_DATA5 12
#define MESSAGE_DATA6 13
#define MESSAGE_DATA7 14

// PDU Format
#define PROPRIETARY_PDU1 239
    // PDU Specific
    // Destination Address
        // Data byte 0
        #define CLEAR_ERRORS 0
        #define CLEAR_WARNINGS 1

// PDU Format
```

```
#define PROPRIETARY_PDU2 255
    // PDU Specific
    #define IM_TIME_DATE              0
    #define IM_FIRMWARE_DATA_PACKET    1
    #define WD_CAPABILITY_UPGRADE_BYTE 2
    #define SHORT_COMMANDS            3 // the command is in byte 0, and data specific to the
command can be placed in the other bytes.
        // Data byte 0
        #define IM_NEW_MOTOR_CURRENT_LIMIT      2
        #define IM_NEW_BATTERY_CURRENT_LIMIT    3
        #define IM_NEW_MOTOR_VOLTAGE_LIMIT      4
        #define IM_SET_INSTALLED_THROTTLE       5
        #define WD_NO_WARNINGS                 11
        #define WD_NO_ERRORS                   12
        #define IM_NEW_MOTOR_CURRENT_RAMP       13
        #define SPEED_SENSORS                  14

    #define WD_POWER_DATA             4
    #define WD_SENSOR_RELATED         5
    #define WD_SUPPLY_VOLTAGES        6
    #define WD_DTC_WARNING            7
    #define WD_DTC_ERROR              8
    #define WD_LIMITS_VOLTAGE_CURRENT  9
    #define WD_SYSTEM_INFO_A          10
    #define WD_SYSTEM_INFO_B          11
    #define WD_LIMITS_VOL_CUR_REV      12


int level_1_tx_buffer[NUM_BUFFER_BYTES];
int level_0_tx_buffer[TX_BUFFER_SIZE][NUM_BUFFER_BYTES];
int tx_buffer_next_in;
int tx_buffer_next_out;
int tx_queue_count;

int level_1_rx_buffer[NUM_BUFFER_BYTES];
int level_0_rx_buffer[RX_BUFFER_SIZE][NUM_BUFFER_BYTES];
int rx_buffer_next_in;
int rx_buffer_next_out;
int rx_queue_count;

int1 tx_buffer_full;
int1 rx_buffer_full;

int1 received_messages_have_been_dropped; // used to detect if messages are being dropped due to a
full rx buffer

// Messages to listen to:
// 1: Addressed to me (0-239 in protocol format and my address in protocol specific)
// 2: Broadcasted (240-255 in protocol format)
```

```
void set_wdp_filters(int device_address) {
    can_set_mode(CAN_OP_CONFIG);

    can_set_id(RX0MASK, 0x10FF00, CAN_USE_EXTENDED_ID);  //set mask 0 (the bit of interest
between 239 and less and 240 and greater is bit 5 (must be 0)
    can_set_id(RX0FILTER0, (int16)device_address << 8, 1);  //set filter 0 of mask 0
    can_set_id(RX0FILTER1, (int16)device_address << 8, 1);  //set filter 1 of mask 0


    can_set_id(RX1MASK, 0x100000, CAN_USE_EXTENDED_ID);  //set mask 1
    can_set_id(RX1FILTER2, 0x100000, 1);  //set filter 0 of mask 1, if bit 5 of protocol format is a 1,
then protocol format is 240-255 and should be listened to
    can_set_id(RX1FILTER3, 0x100000, 1);  //set filter 1 of mask 1
    can_set_id(RX1FILTER4, 0x100000, 1);  //set filter 2 of mask 1
    can_set_id(RX1FILTER5, 0x100000, 1);  //set filter 3 of mask 1

    can_set_mode(CAN_OP_NORMAL); // CAN_OP_LOOPBACK for testing purposes
}

void wdp_init(int initial_address) {
    can_init();
    // set our own acceptance masks and filters
    set_wdp_filters(initial_address);

    // initialize the transmit buffer variables
    tx_buffer_next_in = tx_buffer_next_out = 0;
    tx_buffer_full = FALSE;
    tx_queue_count = 0;

    // initialize the receive buffer variables
    rx_buffer_next_in = rx_buffer_next_out = 0;
    rx_buffer_full = FALSE;
    rx_queue_count = 0;

    received_messages_have_been_dropped = FALSE;
}

// Enqueue a message to be transmitted on the CAN bus
// Data is taken from the level 1 buffer, filled by the application.
// returns status of the requested operation
// 0: queue full
// 1: successfully enqueued
int wdp_enqueue() {
    int i;
    int temp_index;

    if(tx_buffer_full) return QUEUE_FULL;

    for(i=0;i<(NUM_OVERHEAD_BYTES +
```

```
level_1_tx_buffer[MESSAGE_NUM_DATA_BYTES]);i++) level_0_tx_buffer[tx_buffer_next_in][i] =
level_1_tx_buffer[i];

    temp_index=tx_buffer_next_in;
    tx_buffer_next_in=(tx_buffer_next_in+1) % TX_BUFFER_SIZE;

    tx_queue_count++;

    // check to see if the buffer is full now
    if(tx_buffer_next_in==tx_buffer_next_out) {
        tx_buffer_next_in=temp_index;
        tx_buffer_full = TRUE;
    }
    return SUCCESS;
}

// pull a message out of the rx buffer
// returns status of the requested operation
// 0: queue empty
// 1: successfully dequeued a message
int wdp_dequeue() {
    int i;

    if(rx_queue_count > 0) { // this should be checked by the calling application, but verify here
        for(i=0;i<level_0_rx_buffer[rx_buffer_next_out][MESSAGE_NUM_DATA_BYTES]
+NUM_OVERHEAD_BYTES;i++) level_1_rx_buffer[i] = level_0_rx_buffer[rx_buffer_next_out][i];

        rx_buffer_next_out=(rx_buffer_next_out+1) % RX_BUFFER_SIZE;
        rx_queue_count--;

        rx_buffer_full = FALSE; // it can't be full if we just removed something
        return 1;
    }
    return 0;
}

// produce the 29-bit ID (package in a 32 bit number) from the given message info
int32 form_id() {
    int8 high_byte;

    high_byte = level_0_tx_buffer[tx_buffer_next_out][MESSAGE_PRIORITY] << 2;
    high_byte |= (level_0_tx_buffer[tx_buffer_next_out][MESSAGE_RESERVED] << 1);
    high_byte |= level_0_tx_buffer[tx_buffer_next_out][DATA_PAGE];

    return make32(high_byte,level_0_tx_buffer[tx_buffer_next_out][PROTOCOL_FORMAT],
                level_0_tx_buffer[tx_buffer_next_out][PROTOCOL_SPECIFIC],
                level_0_tx_buffer[tx_buffer_next_out][MESSAGE_SOURCE_ADDRESS]);
}
```

```
void parse_id(int32 the_id) {
    level_0_rx_buffer[rx_buffer_next_out][MESSAGE_SOURCE_ADDRESS]=make8(the_id,0);
    level_0_rx_buffer[rx_buffer_next_out][PROTOCOL_SPECIFIC]=make8(the_id,1);
    level_0_rx_buffer[rx_buffer_next_out][PROTOCOL_FORMAT]=make8(the_id,2);
    level_0_rx_buffer[rx_buffer_next_out][DATA_PAGE]=0x01 & make8(the_id,3);
}

// Processes CAN messages, incoming and outgoing.
// Call this as frequently as possible.
// The function will not tie up the processor
// for long in any case, so it is safe to call
// frequently.  Compare to J1939_Poll.
void wdp_poll() {
    int temp_index;
    int32 rx_id;
    int rx_len;
    struct rx_stat rxstat;

    // transmit
    if(tx_queue_count > 0 && can_tbe() ) { // there's data to transmit and room to send it

        if( can_putd(form_id(),
                &level_0_tx_buffer[tx_buffer_next_out][MESSAGE_DATA0], // data bytes beginning
location
                level_0_tx_buffer[tx_buffer_next_out][MESSAGE_NUM_DATA_BYTES], // number
of data bytes
                level_0_tx_buffer[tx_buffer_next_out][MESSAGE_PRIORITY], // priority
                1, // extended ID
                0 // remote frame - not currently using
                )) {
            tx_queue_count--;

            tx_buffer_next_out=(tx_buffer_next_out+1) % TX_BUFFER_SIZE; // update the pointer

            if(tx_buffer_full) tx_buffer_full = FALSE; // we're removing something from the transmit
buffer, so it's no longer full
        }
    }

    // receive
    if( can_kbhit() ) {
        if(rx_buffer_full) received_messages_have_been_dropped = TRUE;
        else {
            can_getd(rx_id, &level_0_rx_buffer[rx_buffer_next_in][MESSAGE_DATA0], rx_len, rxstat);

            parse_id(rx_id); // break the id into its separate pieces
            level_0_rx_buffer[rx_buffer_next_in][MESSAGE_NUM_DATA_BYTES] = rx_len; // get the
number of data bytes
```

```
            rx_queue_count++;

            temp_index=rx_buffer_next_in;
            rx_buffer_next_in=(rx_buffer_next_in+1) % RX_BUFFER_SIZE;

            // check to see if the buffer is full now
            if(rx_buffer_next_in==rx_buffer_next_out) {
                rx_buffer_next_in=temp_index;
                rx_buffer_full = TRUE;
            }
        }
    }
}
```

Sending of data:

```
level_1_tx_buffer[MESSAGE_PRIORITY] = MED_PRIORITY;
level_1_tx_buffer[MESSAGE_RESERVED] = 0;
level_1_tx_buffer[DATA_PAGE] = 0;
level_1_tx_buffer[PROTOCOL_FORMAT] = PROPRIETARY_PDU2;
level_1_tx_buffer[MESSAGE_SOURCE_ADDRESS] = 88;

level_1_tx_buffer[PROTOCOL_SPECIFIC] = WD_POWER_DATA;
level_1_tx_buffer[MESSAGE_NUM_DATA_BYTES] = 8;
level_1_tx_buffer[MESSAGE_DATA0] = make8(motor_amps_averaged,1);
level_1_tx_buffer[MESSAGE_DATA1] = make8(motor_amps_averaged,0);
level_1_tx_buffer[MESSAGE_DATA2] = make8(pack_voltage,1);
level_1_tx_buffer[MESSAGE_DATA3] = make8(pack_voltage,0);
level_1_tx_buffer[MESSAGE_DATA4] = make8(drive,1);
level_1_tx_buffer[MESSAGE_DATA5] = make8(drive,0);
level_1_tx_buffer[MESSAGE_DATA6] = controller_state;
level_1_tx_buffer[MESSAGE_DATA7] = last_halt_reason;

wdp_enqueue(); // get the data in the queue
wdp_poll(); // send out the data in the queue
```

Receiving of data:

```
wdp_poll();

if(rx_queue_count > 0) {
        wdp_dequeue();

        if(level_1_rx_buffer[PROTOCOL_FORMAT] == PROPRIETARY_PDU2) {

                if(level_1_rx_buffer[PROTOCOL_SPECIFIC] == WD_SUPPLY_VOLTAGES ) {
                        supply_dcin_voltage = level_1_rx_buffer[MESSAGE_DATA0];
                        supply_15V_a_voltage = level_1_rx_buffer[MESSAGE_DATA1];
```

```
                        supply_15V_b_voltage = level_1_rx_buffer[MESSAGE_DATA2];
                        supply_5V_a_voltage = level_1_rx_buffer[MESSAGE_DATA3];
                        supply_5V_b_voltage = level_1_rx_buffer[MESSAGE_DATA4];
                }
        }
}
```

Revision History:

2/27/12, RJB: Fixed a mistake in the identifier description where "bytes" was used where "bits" should have been.

8/29/11, RJB

- Fixed error in listing of DTCs related to 15V and 5V power supplies. Added leakage error for WD2.
- Added WD2 addresses.

8/24/11, RJB: Added broadcast message 3.11 for changing leakage options. Also added byte 6 to broadcast message 12 for indicating what leakage option is set in the industrial controller.

8/5/11, RJB: Added MX5 and Lokar throttle units.

7/25/11, RJB: Added in DTC definitions to message 7.

7/7/11, RJB: Added WDT-CP throttle unit.

12/13/10, RJB

- Moved brake and reverse from broadcast message 12 to broadcast message 5.
- Shifted broadcast message 5 bytes down by one byte, to fill the position of the installed throttle byte.
- Moved installed throttle from broadcast message 5 to broadcast message 12.
- Combined active and historical broadcast messages 7 and 8 into a single DTC message, broadcast message 7.
- Added designation of "interval" vs. "non-interval". Non-interval data is data that does not frequently change while interval data can rapidly fluctuate and therefore needs to be transmitted on a regular basis.
- Changed cycle time of interval data to 100ms instead of 400ms.

11/29/10, RJB: Broadcast messages 7 and 8 changed to transmit 4 bytes each, containing the DTCs which are bit representations of the error codes. Previously, each byte contained a single DTC. Messages 3.11 and 3.12 are now available as the no error message is not required.

11/15/10, RJB: Corrected source address on broadcast messages 4-12 from "1, others" to "88".

9/28/10, RJB: Prior to this date, all data transmitted from the controller with multiple bytes was sent low byte first, then high byte. All received data was accepted in the opposite order, high byte first, then low byte. Now, all data containing multiple bytes is transmitted high byte(s) first, low byte(s) last.