
J1939 C Library for CAN-Enabled PICmicro[®] Microcontrollers

*Author: Kim Otten
Kim Otten Software Consulting*

*Co-Author: Caio Gübel
Microchip Technology Inc.*

INTRODUCTION

J1939 is a series of SAE recommended practices that have been developed to provide a standard architecture by which multiple electronic systems on a vehicle can communicate. It was developed by the Truck and Bus Control and Communications Network Subcommittee of the Truck and Bus Electrical and Electronics Committee, but its use is not limited to truck and bus applications. J1939 has been implemented in a broad range of vehicles and transportation systems.

J1939 provides a communication protocol over a CAN network. The CAN network is comprised of two or more interconnected Electronic Control Units (ECUs). As per the SAE J1939-11 specification, the ECUs are connected using linear shielded twisted pair wiring, with a data rate of 250 Kbits/second.

Microchip's CAN-enabled PICmicro devices provide a powerful, flexible and low-cost means of implementing the J1939 protocol on a wide variety of electronic vehicle components. The J1939 C library handles the majority of network management aspects of J1939, allowing the user to concentrate on the primary application. The library provides support for all J1939 address configurations and is easily configurable through Microchip's Application Maestro[™] utility.

For details about the PIC18 family of microcontrollers, refer to one of the CAN-enabled PICmicro microcontroller data sheets, such as the "*PIC18FXX8 Data Sheet*" (DS41159), the "*PIC18F6585/8585/6680/8680 Data Sheet*" (DS30491), the "*PIC18F2585/2680/4585/4680 Data Sheet*" (DS39625) and the "*PICmicro[®] 18C MCU Family Reference Manual*" (DS39500). For details about the J1939 specification, refer to the various SAE J1939 specifications, or visit <http://www.sae.org>.

J1939 OVERVIEW

J1939 is a communication protocol that is implemented on a CAN network. Each Controller Application (CA) has an associated NAME and ADDRESS. The NAME value consists of 8 bytes with the following format:

TABLE 1: CONTROLLER APPLICATION NAME/ADDRESS FORMAT

Arbitrary Address Capable	Industry Group	Vehicle System Instance	Vehicle System	Reserved	Function	Function Instance	ECU Instance	Manufacturer Code	Identity Number
1 Bit	3 Bits	4 Bits	7 Bits	1 Bit	8 Bits	5 Bits	3 Bits	11 Bits	21 Bits

These bytes embed the following information.

Arbitrary Address Capable – If the CA is Arbitrary Address Capable (see the “**J1939 Address Configuration**” section for details).

Industry Group – A 3-bit field that indicates an industry group. These values are specified in SAE J1939.

Vehicle System Instance – This 4-bit field identifies one particular occurrence of a given vehicle system in a given network. If only one instance of a certain vehicle system exists in a network, then this field must be set to ‘0’ to define it as the first instance.

Vehicle System – This 7-bit field defines a group of functions in a network. These values are specified in SAE J1939.

Reserved – This field is reserved for future use by SAE.

Function – This 8-bit field defines the primary function of the CA. These values are specified in SAE J1939.

Function Instance – This 5-bit field identifies the particular occurrence of a given function in a vehicle system and given network. If only one instance of a certain Function exists in a network, then this field must be set to ‘0’ to define it as the first instance

ECU Instance – This 3-bit field is used when multiple ECUs are involved in performing a single function. If only one ECU is used for a particular CA, then this field must be set to ‘0’ to define it as the first instance.

Manufacturer Code – This 11-bit field indicates the manufacturer in charge of the production of the electronic control module. This field is assigned by the committee and is independent of other fields. These values are specified in SAE J1939.

Identity Number – This 21-bit field is assigned by the manufacturer of the ECU. It must be unique and must not change with removal of power. The manufacturer may choose to encode information within this field, but they must also ensure uniqueness.

J1939 Address Configuration

Most CAs on a J1939 network will have a preferred address that will be used based on the CA's primary function. These addresses are defined in the SAE J1939 specification. As a general rule, after power-up, a J1939 module will try to claim its preferred address. If a conflict arises between modules, there are several resolution methods based on the address configuration of the CA. If the CA cannot claim an address, it will not be able to send messages.

Single Address Capable CA – Single Address Capable CAs are not able to change their address independently of external action. There are four different types of Single Address Capable CAs:

- **Non-Configurable Address CA** – This type of CA can have its address changed only by the replacement of the firmware installed in the system.
- **Service Configurable Address CA** – This type of CA may have its source address changed in the field, either by a technician using a proprietary technique, or by another CA sending it a Commanded Address message while the CA is in a “service” mode of operation, which normally requires some sort of external tool/device.
- **Command Configurable Address CA** – This type of CA can have its source address changed in the field through a Commanded Address message sent by another CA without the use of a service tool.
- **Self-Configurable Address CA** – This type of CA may select its own address within a limited set of source addresses based on internal algorithms. Once the CA selects which address it should use, it is not capable of changing its address in the case of a conflict.

Arbitrary Address Capable CA – This type of CA can select its source address from any appropriate source address, including those in the 128-247 range, using an internal algorithm to calculate the address. In case of conflict, it can recalculate and reclaim a new address unless all available addresses are already claimed.

J1939 Messages

J1939 messages are sent using the CAN Extended Frame. A J1939 message consists of the following components:

Priority – This 3-bit field is used to define the priority during arbitration. '000' is the highest priority and is usually associated with high-speed control messages. Low priority is used for noncritical configuration and information messages.

Data Page – This 1-bit field defines on which data page (0 or 1) the message is defined in the J1939 specification. Page 0 contains the messages that are presently defined, while Page 1 is for future expansion.

Protocol Data Unit (PDU) Format (PF) – This 8-bit field determines the format of the message and is one of the fields that determines the Parameter Group Number of the message (see the “**Parameter Group Number**” section). If the value is between 0 and 239, the message is a PDU 1 Format message. These messages are sent to specific addresses (CAs). If the value is between 240 and 255, the message is a PDU 2 Format message. These messages are not sent to a specific address (CA), but are instead broadcast to the entire network.

PDU Specific (PS) – This 8-bit field is either the Destination Address (PDU 1 Format) or the Group Extension (PDU 2 Format).

Source Address – This 8-bit field is the address of the CA that sent the message.

Data Length – The number of data bytes in the message.

Data – Up to 8 bytes of data.

The first five items are placed into the CAN 29-bit extended identifier in the format shown in Table 2.

Messages with more than 8 bytes of data must be sent using the special Broadcast Announce Message (BAM) Transport Protocol.

Most messages are intended to be broadcast messages, or PDU 2 Format, where the message is not sent to a particular address. The J1939 specification defines PDU Format and PDU Specific values for many messages by specifying the message Parameter Group Numbers (see the “**Parameter Group Number**” section).

A J1939 node can send messages to other nodes, or it can request messages from other nodes, either globally or with a specific destination address. If a node receives a request sent to it specifically, it must process the message and send some sort of acknowledgement. If a node receives a global request, it must respond if it can. If a node receives a broadcast message, it must determine whether or not it is relevant.

TABLE 2: J1939 MESSAGE FORMAT

S O F	Identifier 11 Bits											S R R	I D E	Identifier Extension 18 Bits																		R T R
S O F	Priority			R	D P	PDU Format (PF) 6 Bits (MSB)						S R R	I D E	PF (cont.)		PDU Specific (PS) (Destination Address, Group Extension or Proprietary)								Source Address								R T R
	3	2	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					
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	23	25	26	27	28	29	30	31	32	33
	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	

Parameter Group Number

The J1939 specification defines allowable messages by their Parameter Group Number (PGN). The Parameter Group Number is a 3-byte value that uniquely defines the message purpose. A PGN has the following format:

0	Reserved	Data Page Bit	PDU Format	Group Extension
6 bits	1 bit	1 bit	8 bits	8 bits

If the PDU Format value for a message is less than 240, then the last 8 bits of the PGN are set to '0'.

The J1939 specification gives the decimal equivalent of the PGNs. To obtain the PF and PS values to use for a specific message, convert the decimal value from the J1939 specification to hexadecimal and use the last two bytes. These values can then be used to either send messages on the network or to request messages from other CAs (see Example 5A and Example 5B in **Appendix A: "Source Code"**).

LIBRARY OVERVIEW

The J1939 C Library is targeted for use with CAN-enabled PIC18 microcontroller applications written with MPLAB® C18. It offers J1939 communications protocol support for CAs with one of the following address configurations:

- Single Address Capable
 - Non-configurable
 - Service configurable
 - Command configurable
 - Self-configurable
- Arbitrary Address Capable

The library routines handle initialization and network functions automatically. CA messages are placed in a queue for transmission. Messages that are received for the CA are placed in a separate queue for processing. The queue sizes are configurable by the user based on available RAM. Many other aspects of the library operation are also configurable to allow for various hardware and software designs.

The library has some limitations:

- Broadcast Announce Message (BAM) format is explicitly supported only to the extent that the Commanded Address message can be received and processed. Any data transfer packets that are received that are not part of the Commanded Address message are placed into the receive queue for assembly by the CA. The application can send BAM messages, but it must place the individual messages in the transmit queue.
- Working Sets are not explicitly supported.

HOW TO USE THE LIBRARY

The main function of the library is to queue all received messages for processing by the CA, transmit all messages that the CA would like to send and handle all network management transparently to the CA. Received and transmitted messages are stored in their own separate queues, so the interface to the J1939/CAN network is encapsulated in the library. Network management messages, such as address arbitration, are handled entirely by the library, with no additional processing required by the CA. Network management messages also do not require any additional receive or transmit queue space, so the queue size can be customized independently of the library functionality.

PROJECT FILES

The `j1939.c` source file must be compiled and linked with the main CA file(s).

The `j1939.h` header file must be included in any CA source files that utilize the library routines or any J1939 definitions.

These files, along with `j1939.def`, should be located in the project directory. They should not be obtained from a generic directory, since `j1939.def` will change for each CA (the NAME, ADDRESS, etc. will be different).

Basic Setup

The library can be configured either to use the CAN module's interrupt capability or to poll the CAN module.

USING INTERRUPTS

Using interrupts is the preferred method of operation, since it decreases the likelihood that received messages will be missed.

There are three functions that need to be called for basic J1939 support (refer to the **"Library Functions"** section for details):

- `J1939_Initialization`
- `J1939_Poll`
- `J1939_ISR`

After performing any self-test or other initialization, call `J1939_Initialization` to set up the J1939 support and begin the process of establishing the CA's presence on the network. `J1939_Initialization` can either initialize the NAME and ADDRESS values internally, or allow the CA to do it in the case of Self-Configurable Address CAs or CAs that dynamically determine the NAME and ADDRESS values. When this function terminates, the CA will be able to receive global and broadcast messages, but it will not yet have a J1939 address unless the address is in the proprietary address range (less than 128 or between 248 and 253).

After `J1939_Initialization` is called, global interrupts should be enabled by setting the `GIEH/GIE` and/or the `GIEL/PEIE` bits appropriately.

After initialization, call `J1939_Poll` at least every few milliseconds until the `WaitingForAddressClaimContention` flag is clear. When this flag is clear, the CA can check the `CannotClaimAddress` flag. If this flag is clear, then the system is on the network with an established address. If this flag is set, then the CA does not have an accepted address on the network, but the CA can still receive messages sent to the global address and broadcast messages.

If the CA cannot use an address other than its initial address, then it is not necessary to call `J1939_Poll` after `WaitingForAddressClaimContention` is clear. However, if the CA can change its address (either through the `Commanded Address` message or by being `Arbitrary Address Capable`), then call `J1939_Poll` at least every few milliseconds whenever the `WaitingForAddressClaimContention` is set. Note that if the system is configured for interrupts, `J1939_Poll` will not check for received messages or transmit queued messages. If the CA does change its address, global interrupts will be disabled and re-enabled to reconfigure the CAN module. Therefore, `J1939_Poll` should not be called from any place where the configured interrupt enable has been temporarily disabled.

The CA's interrupt handler must call `J1939_ISR` if any of the flags in `PIR3` are set. This function places any received messages into the receive queue for processing and transmits any messages that can be transmitted from the transmit queue. It also automatically handles any network management messages and clears the interrupt flags. No specific error handling is provided other than to clear the interrupt flags. This function can be customized to provide any desired error handling.

USING POLLING

If necessary, the CAN module can be polled to transmit and receive messages. If using polling, use extreme care to ensure the following:

- The CAN module must be polled often enough to avoid missing a received message. The rate of polling depends on the selected CAN bus speed.
- When placing a message in the transmit queue, ensure that the routine, `J1939_Poll`, is called between successive attempts to queue the message; otherwise, an infinite loop will be generated.

There are two functions that need to be called for basic J1939 function support (refer to the “**Library Functions**” section for details):

- `J1939_Initialization`
- `J1939_Poll`

After performing any self-test or other initialization, call `J1939_Initialization` to set up the J1939 support and begin the process of establishing the CA's presence on the network. `J1939_Initialization` can either initialize the `NAME` and `ADDRESS` values internally, or allow the CA to do it in the case of `Self-Configurable Address CAs` or `CAs` that dynamically determine the `NAME` and `ADDRESS` values. When this function terminates, the CA will be able to receive global and broadcast messages, but it will not yet have a J1939 address unless the address is in the proprietary address range (less than 128 or between 248 and 253).

After initialization, call `J1939_Poll` at least every few milliseconds until the `WaitingForAddressClaimContention` flag is clear. When this flag is clear, the CA can check the `CannotClaimAddress` flag. If this flag is clear, then the system is on the network with an established address. If this flag is set, then the CA does not have an accepted address on the network, but the CA can still receive messages sent to the global address and broadcast messages.

Continue to call `J1939_Poll` to transmit any messages that can be transmitted from the transmit queue and place any received messages into the receive queue for processing. Any network management messages are handled automatically.

MESSAGES

Message Definition and Structure

Create one or more message buffers using the following definition:

```
J1939_MESSAGE MyMessage;
```

Since each message buffer requires 13 bytes of RAM, try to keep the number of message buffers to a minimum. In most situations, only one or two CA message buffers will be needed.

The message structure is defined in `j1939.h`, but here are the main fields that the CA will use. Refer to the “**J1939 Messages**” section and the J1939 specification for more details on each portion of the message.

- `MyMessage.DataPage`, one bit
- `MyMessage.Priority`, three bits
- `MyMessage.PDUFormat`, one byte
- `MyMessage.PDUSpecific`, one byte

This field can also be referenced as `DestinationAddress` or `GroupExtension` to help clarify the CA code.

- `MyMessage.SourceAddress`, one byte (this is automatically filled in by the library before transmission).
- `MyMessage.DataLength`, 4 bits, but must be between 0 and 8.
- `MyMessage.Data[8]`, array of 8 bytes. Most messages are sent LSB first.

RECEIVED MESSAGES

Network management messages are handled automatically by the library. Any other messages are queued for processing by the CA. These messages include:

- Broadcast messages
- Messages sent to the CA's address
- Messages sent to the global address

Check the variable, `RXQueueCount`, to see if there are any messages ready in the queue. Call the routine, `J1939_DequeueMessage`, to pull one message out of the receive queue and place it in a buffer for processing. Check the flag, `J1939_Flags.ReceivedMessages-Dropped`, to see if any messages have been dropped. Refer to the description of `J1939_DequeueMessage` for more details.

TRANSMIT MESSAGES

Network management messages are handled automatically by the library. Place any other messages the CA wishes to send into the transmit queue by calling `J1939_EnqueueMessage` to copy the message into the transmit queue. Refer to the description of `J1939_EnqueueMessage` for more details. The routine, `J1939_TransmitMessages`, performs the actual transmission of the message when it is called from either `J1939_Poll` or `J1939_ISR`.

If BAM messages are being sent, call `J1939_EnqueueMessage` with each individual message of the BAM message.

LOSS OF J1939 ADDRESS

If another J1939 node on the bus claims the same address, the two nodes' NAMES are compared. The node with the lower NAME value is allowed to keep the address and the other node must relinquish it. If the latter happens, the CA is no longer allowed to transmit messages, other than the Cannot Claim Address message and it can only receive messages sent to the global address or broadcast messages. This is handled automatically by the library.

If the CA is Arbitrary Address Capable, then it will call the user-defined routine, `CA_RecalculateAddress`, to get another address to try to claim. If it returns a value of `TRUE`, the library will attempt to claim the new address. If it returns a value of `FALSE`, the library will not attempt to claim another address.

USING THE COMMANDED ADDRESS MESSAGE

If the library has been configured to allow reception of the Commanded Address message, then the Commanded Address message will be automatically processed when it is received. The system will initiate a claim to the new address and if successful, use that address for subsequent transmissions. If the claim is unsuccessful, then the CA will no longer be able to transmit messages and can only receive messages sent to the global address or broadcast messages.

If the CA wishes to send the Commanded Address message, it must enqueue the BAM and two DT packets as per the J1939-21 specification. Refer to the following Commanded Address transmission example.

SELF-CONFIGURABLE CA

If the CA is Self-Configurable, it can select its address from more than one possible value. The CA should initialize `J1939_Address` and the `CA_Name` array based on whatever calculations necessary prior to calling `J1939_Initialization`. The CA should then call `J1939_Initialization` with the parameter value of `FALSE`, indicating that the NAME and ADDRESS values have already been initialized.

ARBITRARY ADDRESS CAPABLE CA

If the CA is Arbitrary Address Capable, it can select from more than one address to try to claim. If the library cannot claim an address, it will call the user defined routine, `CA_RecalculateAddress`. This routine can then calculate a new address to try. If all potential addresses have been tried and have failed, this routine should return `FALSE` to indicate that there are no more addresses to try. Otherwise, the routine should return `TRUE` so the library can attempt to claim the new address.

Note that the definition of whether or not the CA is Arbitrary Address Capable is indicated by a bit in one of the NAME bytes. The library code will use the definition as per the Application Maestro definition to conditionally compile the extra code needed for Arbitrary Address Capable CAs, but during address arbitration, the values in the NAME array will be used. If the NAME array is initialized by the CA, ensure that the Arbitrary Address Capable bit is set up correctly.

LIBRARY CONFIGURATION

The library is configured through the use of the Application Maestro utility. This utility will copy all required source files into the specified directory and will generate a single definition file, `j1939.def`, containing definitions that will configure the library operation. The configuration of that file is described here.

In addition, one or two C functions may be required in the CA code, depending on the CA's address capability.

J1939 Configuration

Starting Address – Define the initial CA address value. Note that this address value must be a valid value as per the J1939 specification.

Arbitrary Address Capable – Indicate whether or not the CA is Arbitrary Address Capable. If it is, it must contain a function to calculate a new address to try to claim if the current address claim fails. It must also return whether or not to try to claim the new address. The function must have the prototype:

```
BOOL CA_RecalculateAddress( unsigned char
*NewAddress );
```

Industry Group Number – A 3-bit value indicating the type of industry for which the CA is defined, as per SAE J1939.

Vehicle System Instance – A 4-bit value indicating the particular occurrence of the particular vehicle system within the network.

Vehicle System – A 7-bit value indicating the type of vehicle system, as per SAE J1939.

J1939 Function – An 8-bit value indicating the function of the CA, as per SAE J1939.

Function Instance – A 5-bit value indicating the particular occurrence of the particular function within the network.

ECU Instance – A 3-bit value indicating the particular ECU associated with a given function.

Manufacturer Code – An 11-bit value indicating the company that was responsible for the production of the ECU, as per SAE J1939.

Identity Number – A 21-bit value chosen by the manufacturer to uniquely identify the CA. This field should be unique and non-varying with removal of power. If this value is needed for manual assignment of the NAME array, it can be accessed by the label, `J1939_IDENTITY_NUMBER`.

Commanded Address Message Accepted – If the CA can accept the Commanded Address message. It is important that this parameter is set to "Yes" only when the Commanded Address message can be accepted to allow the library compilation to optimize out any unnecessary functions. Also, the CA must provide a function that returns `TRUE` if the Commanded Address message can be accepted and a `FALSE` if it must be ignored. In the case of a Service Configurable CA, the function may perform another function, such as checking the level on a pin to see if a service tool has been installed. In the case of a Command Configurable CA, the function may simply return `TRUE`. The function must have the prototype:

```
BOOL CA_AcceptCommandedAddress( void );
```

Message Queue Configuration

Receive Queue Size – The number of messages that can be stored in the queue of received messages. This value must be greater than or equal to '1'.

Overwrite Receive Queue – If the receive queue is full and another message is received, this message can either be dropped or can overwrite the previous message. If "Yes" is selected, the new message overwrites the previous message. If "No" is selected, the message is dropped and a flag is set to indicate that received messages have been dropped.

Transmit Queue Size – The number of messages that can be stored in the queue of messages to transmit. This value must be greater than or equal to '1'.

Overwrite Transmit Queue – If the transmit queue is full and another message is queued for transmit, this message can either be dropped or can overwrite the previous message. If "Yes" is selected, the new message overwrites the previous message. If "No" is selected, the message is dropped and a return code indicates that the message was not queued.

Interrupts Versus Polling

Interrupts or Polling – If interrupts are used to service the CAN module or if the CAN module is polled.

Enable Prioritized Interrupts – If interrupts are used, this determines if the interrupts are prioritized. This value is ignored if polling is used. Note that it is important to select this option accurately, as it will set up the RCON register, as well as identify how to enable and disable interrupts in certain sections of the library.

Receive Interrupt Priority – The priority of receive and error interrupts if prioritized interrupts are used. If interrupts or prioritization is not used, this value is ignored.

Transmit Interrupt Priority – The priority of transmit interrupts if prioritized interrupts are used. If interrupts or prioritization is not used, this value is ignored.

ECAN™ (Enhanced CAN) Module Configuration

CAN Module Mode – If an ECAN module-enabled device is used, FIFO mode is recommended. If a CAN-enabled device is chosen, then the Legacy mode must be selected.

Extra Receive Buffers – If FIFO mode is used, how many of the six extra buffers are to be configured as receive buffers. The remaining buffers will be configured as transmit buffers. If Legacy mode is used, this value is ignored.

CAN Bit Timing

The CAN bit timing is set by the following values. Refer to the device data sheet or the *"PICmicro® 18C MCU Family Reference Manual"* for details on defining these values. Ensure that all nodes in the system have identical bit timing by adjusting these values appropriately to account for the different system clocks and hardware designs.

- Synchronized Jump Width x T_Q
- Baud Rate Prescaler
- Phase Segment 2 Time Select
- Sample of CAN Bus Line
- Phase Segment 1 x T_Q
- Propagation Time Select x T_Q
- Wake-up Disable
- CAN Bus Line Filter for Wake-up
- Phase Segment 2 x T_Q

CAN BIT TIMING AND PROCESSOR SPEED

A CAN network can run with a minimum Nominal Bit Time of 1 microsecond. J1939 messages utilize the CAN extended frame. A minimum of 67 bit times are required to transmit one J1939 message with a data length of zero. If the system is operating at full speed, a new message can appear on the bus every 67 microseconds.

The PIC18 family can run with a system clock as slow as 32 kHz. This gives an instruction time of 125 microseconds per instruction. It is obviously not practical to design a system such that two J1939 messages can be received in the time it takes the PIC18 device to execute a single instruction.

Take care that the PIC18 processor speed and CAN network speed are selected such that the processor will have adequate time to process messages. For example, in a worst case scenario, a PIC18 device running at 16 MHz connected to a 1 MHz CAN network would have 268 instruction cycles to process a message. Many messages will contain data bytes and therefore, take longer to receive and the network should not have 100% loading, so in practice, the PIC18 will have more time to process the message. But care should be taken when designing the system that there is adequate buffering in case of message bursts.

LIBRARY FUNCTIONS

Interface Variables

CA ADDRESS

The CA ADDRESS can be accessed through the variable, `J1939_Address`. Unless the CA is Self-Configurable, this value should not be required by the CA. In all cases, the CA *must not* modify this variable after `J1939_Initialization` is called. Messages that are transmitted by using `J1939_EnqueueMessage` automatically have this value inserted into the Source Address portion of the message.

CA NAME

The CA NAME can be accessed and modified, if necessary, through the array, `CA_Name`. This array is an array of unsigned chars, stored Least Significant Byte first.

J1939 STATUS

The network status can be obtained by looking at two variables, `J1939_Flags` and `RXQueueCount`. The following flags in `J1939_Flags` are of use to the CA:

- `J1939_Flags.CannotClaimAddress` – set to '1' if either an address has not yet been claimed or the address cannot be claimed.
- `J1939_Flags.WaitingForAddressClaimContention` – set to '1' if the system is trying to claim an address and is waiting for a claim contention. If this flag is set, `J1939_Poll` must be called every few milliseconds, even if interrupts are being used, to check for contention time-out.
- `J1939_Flags.ReceivedMessagesDropped` – set if overwrite receive queue was set to "Yes" and received messages have been dropped because the queue was full. The CA must clear this flag.

The CA can alter only `J1939_Flags.ReceivedMessagesDropped`. The CA *must not* alter the other flags.

`RXQueueCount` is the number of messages in the receive queue waiting for processing by the CA.

External Interface Routines

The following routines, listed in Table 3, are provided for the developer.

TABLE 3: EXTERNAL INTERFACE ROUTINES

Name	Description
<code>J1939_DequeueMessage</code>	Copy message from receive queue to user buffer
<code>J1939_EnqueueMessage</code>	Copy message from user buffer to transmit queue
<code>J1939_Initialization</code>	Initializes all necessary variables, configures the CAN module and starts the address claim process
<code>J1939_ISR</code>	Interrupt-based reception and transmission routine
<code>J1939_Poll</code>	Polling based reception and transmission routine; also required if the CA is Service Configurable, Command Configurable or Arbitrary Address Capable

FUNCTION DESCRIPTIONS

Function unsigned char J1939_DequeueMessage(J1939_MESSAGE *MsgPtr)
Preconditions System initialized by J1939_Initialization
Overview This function pulls a received message out of the queue and places it into the buffer pointed to by *MsgPtr.
Input J1939_MESSAGE *MsgPtr – A pointer to the message buffer.
Output RC_SUCCESS – Message dequeued successfully.
RC_QUEUEEMPTY – No messages to return.
RC_CANNOTRECEIVE – System cannot currently receive messages. This is returned only after the receive queue is empty.
Side Effects None

Function unsigned char J1939_EnqueueMessage(J1939_MESSAGE *MsgPtr)
Preconditions System initialized by J1939_Initialization
Overview This function takes the message in the CA's RAM pointed to by *MsgPtr and places it in the queue for transmission. The message will automatically have the CA's Source Address placed in the proper field, but the CA must fill in the other values.
Input Message pointed to by *MsgPtr
Output RC_SUCCESS – Message enqueued successfully.
RC_QUEUEFULL – Transmit queue full; message not queued.
RC_CANNOTTRANSMIT – System cannot currently transmit messages.
Side Effects None

Function void J1939_Initialization(BOOL InitNAMEandAddress)
Preconditions None
Overview This function must be called after any CA self-test and basic initialization. It initializes the library's global variables, the CAN module and interrupts, if necessary. It then initiates the process of establishing the CA's address on the network.
Input BOOL InitNAMEandAddress – Flag: If InitNAMEandAddress is TRUE, it will initialize the CA's J1939 NAME and ADDRESS to the values selected through the Application Maestro utility. If it is FALSE, it will not initialize these values, allowing the user to dynamically define the NAME and initial ADDRESS.
Output None
Side Effects None

Function void J1939_ISR(void)
Preconditions System initialized by J1939_Initialization
Overview The CA must call this function if it receives an interrupt and one or more of the flags in PIR3 is set. This function calls J1939_ReceiveMessage to process any received messages, J1939_TransmitMessage to transmit any messages in the transmit queue and checks for any errors. It then clears the appropriate flags.
Input None
Output None
Side Effects None

Function	<code>void J1939_Poll(unsigned long ElapsedTime)</code>
Preconditions	System Initialized by J1939_Initialization
Overview	<p>After J1939_Initialization, this function must be called at least every few milliseconds, with the elapsed time in microseconds, until J1939_Flags.WaitingForAddressClaimContention is clear in order to establish that there is no contention for the CA's address on the network. If interrupts are not used, this function must also be called at least every few milliseconds during the CA's functioning to check for received messages. If interrupts are used and either the Commanded Address message can be accepted or the CA is Arbitrary Address Capable, then this function must still be called at least every few milliseconds during the CA's main processing to check for address contention in response to claiming a new address. If interrupts are used, then J1939_Poll will not check for received messages or messages to transmit, but will allow the J1939_ISR to handle that processing.</p>
Input	<code>unsigned long ElapsedTime</code> – This value can be a value calculated at run time or a constant value. A precise value of <code>ElapsedTime</code> is not necessary, but the value should not be any greater than the actual elapsed time to ensure that the minimum 250 ms contention wait time is met.
Output	None
Side Effects	If interrupts are being used, this routine may enable global interrupts. Therefore, this routine should be called only when global interrupts are enabled. If interrupts are not being used, this routine will not enable global interrupts.

CONCLUSION

This library provides the J1939 network management functionality necessary to implement J1939 protocol on an ECU. The library is easily configurable through the Application Maestro utility and supports all J1939 addressing configurations. Its simple interface and ease of configuration allow the developer to add J1939 compatibility to his application, while maintaining focus on the primary ECU functionality.

APPENDIX A: SOURCE CODE

Software License Agreement

The software supplied herewith by Microchip Technology Incorporated (the "Company") is intended and supplied to you, the Company's customer, for use solely and exclusively with products manufactured by the Company.

The software is owned by the Company and/or its supplier, and is protected under applicable copyright laws. All rights are reserved. Any use in violation of the foregoing restrictions may subject the user to criminal sanctions under applicable laws, as well as to civil liability for the breach of the terms and conditions of this license.

THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.

The source code, with all required support files, is available for download as a Zip archive from the Microchip web site, at:

www.microchip.com

Examples

The following examples show how the J1939 library routines are used in a CA. Note that the applications and values are for demonstration only and are not intended to mimic an actual automotive application.

Read these examples in order, as each one builds on the last. Important items to note from one example to the next are shown in **bold** text.

These examples can be run on a PICDEM™ CAN LIN 3 Demonstration Board. Set the configuration bits to the following settings (the remaining configuration bits may be left at their default values):

- Oscillator: HS
- Brown-out Detect: Disabled
- Watchdog Timer: Disabled
- Low-Voltage Program: Disabled

EXAMPLE 1:

```
/*
*****
*****
Example 1

This example shows a very simple J1939 implementation. It uses polling
to check for a message to light an LED and to send a message if a
button is pressed.

Both Node 0 and Node 1 should be programmed with the same code, except
that OTHER_NODE should be defined as the other node's J1939 Address.

Application Maestro should be run with the following options changed
from their default values (in addition to NAME, Address, and bit rate
values):

Interrupts or Polling - Polling
*/

#include <p18cxxx.h>
#include "j1939.h"

J1939_MESSAGE Msg;

// Define some arbitrary values. They must agree with the other node's
// values.

#define OTHER_NODE          129
#define TURN_ON_LED         92
#define TURN_OFF_LED        94

void main( void )
{
```

```

unsigned char    LastSwitch = 1;
unsigned char    CurrentSwitch;

TRISBbits.TRISB4 = 1;           // Switch pin
TRISD = 0;                     // LED pins
LATD = 0;                       // Turn off LED

J1939_Initialization( TRUE );

// Wait for address contention to time out
while (J1939_Flags.WaitingForAddressClaimContention)
    J1939_Poll(5);

// Now we know our address should be good, so start checking for
// messages and switches.

while (1)
{
    CurrentSwitch = PORTBbits.RB4;
    if (LastSwitch != CurrentSwitch)
    {
        Msg.DataPage           = 0;
        Msg.Priority            = J1939_CONTROL_PRIORITY;
        Msg.DestinationAddress  = OTHER_NODE;
        Msg.DataLength          = 0;
        if (CurrentSwitch == 0)
            Msg.PDUFormat = TURN_ON_LED;
        else
            Msg.PDUFormat = TURN_OFF_LED;
        while (J1939_EnqueueMessage( &Msg ) != RC_SUCCESS)
            J1939_Poll(5);
        LastSwitch = CurrentSwitch;
    }

    while (RXQueueCount > 0)
    {
        J1939_DequeueMessage( &Msg );
        if (Msg.PDUFormat == TURN_ON_LED)
            LATDbits.LATD0 = 1;
        else if (Msg.PDUFormat == TURN_OFF_LED)
            LATDbits.LATD0 = 0;
    }

    // Since we don't accept the Commanded Address message,
    // the value passed here doesn't matter.
    J1939_Poll(20);
}
}

```

```

TRISBbits.TRISB4 = 1;           // Switch pin
TRISD = 0;                     // LED pins
LATD = 0;                      // Turn off LED

J1939_Initialization( TRUE );
INTCONbits.PEIE = 1; // Enable peripheral interrupts
INTCONbits.GIE = 1;  // Enable global interrupts

// Wait for address contention to time out
while (J1939_Flags.WaitingForAddressClaimContention)
    J1939_Poll(5);

// Now we know our address should be good, so start checking for
// messages and switches.

while (1)
{
    CurrentSwitch = PORTBbits.RB4;
    if (LastSwitch != CurrentSwitch)
    {
        Msg.DataPage           = 0;
        Msg.Priority           = J1939_CONTROL_PRIORITY;
        Msg.DestinationAddress = OTHER_NODE;
        Msg.DataLength         = 0;
        if (CurrentSwitch == 0)
            Msg.PDUFormat = TURN_ON_LED;
        else
            Msg.PDUFormat = TURN_OFF_LED;

        // We don't need to call J1939_Poll in the middle of this
        // loop, since the queue will be emptied during interrupt
        // processing.
        while (J1939_EnqueueMessage( &Msg ) != RC_SUCCESS);

        LastSwitch = CurrentSwitch;
    }

    while (RXQueueCount > 0)
    {
        J1939_DequeueMessage( &Msg );
        if (Msg.PDUFormat == TURN_ON_LED)
            LATDbits.LATD0 = 1;
        else if (Msg.PDUFormat == TURN_OFF_LED)
            LATDbits.LATD0 = 0;
    }

    // We don't need to call J1939_Poll, since the queues will
    // be managed during the interrupt processing.
}
}

```

```
#define SECOND_ADDRESS      132
```

```

// High priority interrupt vector

```

High priority, immediate routine

#pragma omp critical { printf("Hello\n"); }


```

{
    if (PIR3 != 0x00)
        J1939_ISR();
}

//-----

void main( void )
{
    unsigned char    LastSwitch = 1;
    unsigned char    CurrentSwitch;
    unsigned char    PushCount = 0;

    TRISBbits.TRISB4 = 1;           // Switch pin
    TRISD = 0;                     // LED pins
    LATD = 0;                       // Turn off LED

    J1939_Initialization( TRUE );
    INTCONbits.GIEH = 1;

    // Wait for address contention to time out
    while (J1939_Flags.WaitingForAddressClaimContention)
        J1939_Poll(5);

    // Now we know our address should be good, so start checking for
    // messages and switches.

    while (1)
    {
        CurrentSwitch = PORTBbits.RB4;
        if (LastSwitch != CurrentSwitch)
        {
            Msg.DataPage          = 0;
            Msg.Priority           = J1939_CONTROL_PRIORITY;
            Msg.DestinationAddress = SECOND_ADDRESS;
            Msg.DataLength         = 0;
            if (CurrentSwitch == 0)
                Msg.PDUFormat = TURN_ON_LED;
            else
            {
                Msg.PDUFormat = TURN_OFF_LED;
                if (PushCount < 6)
                    PushCount ++;
            }
        }
        while (J1939_EnqueueMessage( &Msg ) != RC_SUCCESS);
        LastSwitch = CurrentSwitch;

        if (PushCount == 5)
        {
            Msg.DataPage          = 0;
            Msg.Priority           = J1939_TP_CM_PRIORITY;
            Msg.DestinationAddress = J1939_GLOBAL_ADDRESS;
            Msg.DataLength         = 8;
            Msg.PDUFormat          = J1939_PF_TP_CM;
            Msg.Data[0]            = J1939_BAM_CONTROL_BYTE;
            Msg.Data[1]            = 9;    // 9 data bytes
            Msg.Data[2]            = 0;
            Msg.Data[3]            = 2;    // 2 packets
            Msg.Data[4]            = 0xFF; // Reserved
            Msg.Data[5]            = 0xD8; // PGN
            Msg.Data[6]            = 0xFE; // PGN
            Msg.Data[7]            = 0x00; // PGN
            while (J1939_EnqueueMessage( &Msg ) != RC_SUCCESS);

            Msg.DataPage          = 0;

```

```
Msg.Priority           = J1939_TP_DT_PRIORITY;
Msg.DestinationAddress = J1939_GLOBAL_ADDRESS;
Msg.DataLength         = 8;
Msg.PDUFormat          = J1939_PF_DT;
Msg.Data[0]            = 1;    // First packet
Msg.Data[1]            = NODE_NAME0;
Msg.Data[2]            = NODE_NAME1;
Msg.Data[3]            = NODE_NAME2;
Msg.Data[4]            = NODE_NAME3;
Msg.Data[5]            = NODE_NAME4;
Msg.Data[6]            = NODE_NAME5;
Msg.Data[7]            = NODE_NAME6;
while (J1939_EnqueueMessage( &Msg ) != RC_SUCCESS);

Msg.Data[0]            = 2;    // Second packet
Msg.Data[1]            = NODE_NAME7;
Msg.Data[2]            = SECOND_ADDRESS;
Msg.Data[3]            = 0xFF;
Msg.Data[4]            = 0xFF;
Msg.Data[5]            = 0xFF;
Msg.Data[6]            = 0xFF;
Msg.Data[7]            = 0xFF;
while (J1939_EnqueueMessage( &Msg ) != RC_SUCCESS);
}

}

while (RXQueueCount > 0)
{
    J1939_DequeueMessage( &Msg );
    if (Msg.PDUFormat == TURN_ON_LED)
        LATDbits.LATD0 = 1;
    else if (Msg.PDUFormat == TURN_OFF_LED)
        LATDbits.LATD0 = 0;
}
}

}
```

EXAMPLE 3B:

```

/*****
*****/

```

```

/*

```

```

Example 3b

```

This example shows what the receiving node for Example 3a should look like, using the same concept as Example 2 of using interrupts to check for a message to light an LED and to send a message if a button is pressed. Note that three basic changes are required:

- it must accept the Commanded Address message (Application Maestro)
- it must have a CA_AcceptCommandedAddress function
- it must call J1939_Poll during the main loop, even though interrupts are being used.

The rest of the code is identical. The change of address will be handled in the background.

Application Maestro should be run with the following options changed from their default values (in addition to NAME, Address, and bit rate values):

Commanded Address Message Accepted - Yes

```

*/

```

```

#include <p18cxxx.h>

```

```

#include "j1939.h"

```

```

J1939_MESSAGE Msg;

```

```

// Define some arbitrary values. They must agree with the other node's
// values.

```

```

#define OTHER_NODE          129
#define TURN_ON_LED         92
#define TURN_OFF_LED        94
#define MAIN_LOOP_TIME_IN_MICROSECONDS 100

```

```

BOOL CA_AcceptCommandedAddress( void )
{
    return 1;
}

```

```

void InterruptHandlerLow (void);

```

```

//-----
// Low priority interrupt vector

```

```

#pragma code InterruptVectorLow = 0x0018
void InterruptVectorLow( void )
{
    _asm
        goto InterruptHandlerLow
    _endasm
}

```

```

//-----
// Low priority interrupt routine

```

```

#pragma code
#pragma interruptlow InterruptHandlerLow

```

```

void InterruptHandlerLow( void )
{
    if (PIR3 != 0x00)
        J1939_ISR();
}

```

```
}

//-----

void main( void )
{
    unsigned char    LastSwitch = 1;
    unsigned char    CurrentSwitch;

    TRISBbits.TRISB4 = 1;           // Switch pin
    TRISD = 0;                     // LED pins
    LATD = 0;                       // Turn off LED

    J1939_Initialization( TRUE );
    INTCONbits.PEIE = 1; // Enable peripheral interrupts
    INTCONbits.GIE = 1; // Enable global interrupts

    // Wait for address contention to time out
    while (J1939_Flags.WaitingForAddressClaimContention)
        J1939_Poll(5);

    // Now we know our address should be good, so start checking for
    // messages and switches.

    while (1)
    {
        CurrentSwitch = PORTBbits.RB4;
        if (LastSwitch != CurrentSwitch)
        {
            Msg.DataPage          = 0;
            Msg.Priority           = J1939_CONTROL_PRIORITY;
            Msg.DestinationAddress = OTHER_NODE;
            Msg.DataLength         = 0;
            if (CurrentSwitch == 0)
                Msg.PDUFormat = TURN_ON_LED;
            else
                Msg.PDUFormat = TURN_OFF_LED;
            while (J1939_EnqueueMessage( &Msg ) != RC_SUCCESS);
            LastSwitch = CurrentSwitch;
        }

        while (RXQueueCount > 0)
        {
            J1939_DequeueMessage( &Msg );
            if (Msg.PDUFormat == TURN_ON_LED)
                LATDbits.LATD0 = 1;
            else if (Msg.PDUFormat == TURN_OFF_LED)
                LATDbits.LATD0 = 0;
        }

        // We need to call J1939_Poll since we can accept the
        // Commanded Address message. Now the time value passed in
        // is important.
        J1939_Poll( MAIN_LOOP_TIME_IN_MICROSECONDS );
    }
}
```

EXAMPLE 4:

```

/*****
/*****

/*
Example 4

This example shows the same concept as Example 2, except that broadcast messages
are used rather than messages sent to a specific address.

Both Node 0 and Node 1 should be programmed with the same code, except
that OTHER_NODE should be defined as the other node's J1939 Address.

Application Maestro should be run with the following options changed from their
default values (in addition to NAME, Address, and bit rate values):

None
*/

#include <p18cxxx.h>
#include "j1939.h"

J1939_MESSAGE Msg;

// Define some arbitrary values. They must agree with the other node's
// values.

#define OTHER_NODE          129
#define TURN_ON_LED         4
#define TURN_OFF_LED        5

void InterruptHandlerLow (void);

//-----
// Low priority interrupt vector

#pragma code InterruptVectorLow = 0x0018
void InterruptVectorLow( void )
{
    _asm
        goto InterruptHandlerLow
    _endasm
}

//-----
// Low priority interrupt routine

#pragma code
#pragma interruptlow InterruptHandlerLow

void InterruptHandlerLow( void )
{
    if (PIR3 != 0x00)
        J1939_ISR();
}

//-----

```

```
void main( void )
{
    unsigned char    LastSwitch = 1;
    unsigned char    CurrentSwitch;

    TRISBbits.TRISB4 = 1;           // Switch pin
    TRISD = 0;                     // LED pins
    LATD = 0;                       // Turn off LED

    J1939_Initialization( TRUE );
    INTCONbits.PEIE = 1; // Enable peripheral interrupts
    INTCONbits.GIE = 1; // Enable global interrupts

    // Wait for address contention to time out
    while (J1939_Flags.WaitingForAddressClaimContention)
        J1939_Poll(5);

    // Now we know our address should be good, so start checking for
    // messages and switches.

    while (1)
    {
        CurrentSwitch = PORTBbits.RB4;
        if (LastSwitch != CurrentSwitch)
        {
            Msg.DataPage           = 0;
            Msg.Priority            = J1939_CONTROL_PRIORITY;
            Msg.DestinationAddress  = OTHER_NODE;
            Msg.PDUFormat           = 254;
            Msg.DataLength          = 0;
            if (CurrentSwitch == 0)
                Msg.GroupExtension= TURN_ON_LED;
            else
                Msg.GroupExtension= TURN_OFF_LED;
            while (J1939_EnqueueMessage( &Msg ) != RC_SUCCESS);
            LastSwitch = CurrentSwitch;
        }

        while (RXQueueCount > 0)
        {
            J1939_DequeueMessage( &Msg );
            if (Msg.GroupExtension == TURN_ON_LED)
                LATDbits.LATD0 = 1;
            else if (Msg.GroupExtension == TURN_OFF_LED)
                LATDbits.LATD0 = 0;
        }
    }
}
```

EXAMPLE 5A:

```

/*****
*****/

```

```

/*

```

```

Example 5a

```

This example shows what a request and acknowledge sequence might look like. This node reads the value of a potentiometer on an analog input pin. When it receives a request for Engine Speed, it will check a switch value. If the switch is pressed, the value of the potentiometer will be returned as the engine speed. If the switch is not pressed, a NACK response will be sent.

Application Maestro should be run with the following options changed from their default values (in addition to NAME, Address, and bit rate values):

```

None

```

```

*/

```

```

#include <p18cxxx.h>

```

```

#include "j1939.h"

```

```

J1939_MESSAGE Msg;

```

```

void InterruptHandlerLow (void);

```

```

//-----
// Low priority interrupt vector

```

```

#pragma code InterruptVectorLow = 0x0018

```

```

void InterruptVectorLow( void )

```

```

{

```

```

    _asm

```

```

        goto InterruptHandlerLow

```

```

    _endasm

```

```

}

```

```

//-----
// Low priority interrupt routine

```

```

#pragma code

```

```

#pragma interruptlow InterruptHandlerLow

```

```

void InterruptHandlerLow( void )

```

```

{

```

```

    if (PIR3 != 0x00)

```

```

        J1939_ISR();

```

```

}

```

```

//-----
#define J1939_PGN0_REQ_ENGINE_SPEED          0x04
#define J1939_PGN1_REQ_ENGINE_SPEED          0xF0
#define J1939_PGN2_REQ_ENGINE_SPEED          0x00

```

```

void main()

```

```

{

```

```

    unsigned char    EngineSpeed;

```

```

    unsigned char    Temp;

```

```

    RCONbits.IPEN = 1;

```

```

    TRISAbits.TRISA5 = 1;

```

```

    TRISBbits.TRISB4 = 1;

```

```

    TRISD = 0;

```

```

    LATD = 0;

```

```

    ADCON0 = 0b00101001;

```

```
ADCON1 = 0x00;

J1939_Initialization( TRUE );
INTCONbits.PEIE = 1;
INTCONbits.GIE = 1;

// Wait for address contention to time out
while (J1939_Flags.WaitingForAddressClaimContention)
    J1939_Poll(5);

// Now we know our address is good, so start checking for
// messages and switches.

while (1)
{
    for (Temp=0; Temp<100; Temp++);
    ADCON0bits.GO = 1;
    while (ADCON0bits.DONE);
    EngineSpeed = ADRESH;

    LATD = EngineSpeed;

    while (RXQueueCount > 0)
    {
        J1939_DequeueMessage( &Msg );
        if (Msg.PDUFormat == J1939_PF_REQUEST)
        {
            if ((Msg.Data[0] == J1939_PGN0_REQ_ENGINE_SPEED) &&
                (Msg.Data[1] == J1939_PGN1_REQ_ENGINE_SPEED) &&
                (Msg.Data[2] == J1939_PGN2_REQ_ENGINE_SPEED))
            {
                if (PORTBbits.RB4)
                {
                    Msg.Priority           = J1939_ACK_PRIORITY;
                    Msg.DataPage           = 0;
                    Msg.PDUFormat          = J1939_PF_ACKNOWLEDGMENT;
                    Msg.DestinationAddress = Msg.SourceAddress;
                    Msg.DataLength         = 8;
                    Msg.Data[0]           = J1939_NACK_CONTROL_BYTE;
                    Msg.Data[1]           = 0xFF;
                    Msg.Data[2]           = 0xFF;
                    Msg.Data[3]           = 0xFF;
                    Msg.Data[4]           = 0xFF;
                    Msg.Data[5]           = J1939_PGN0_REQ_ENGINE_SPEED;
                    Msg.Data[6]           = J1939_PGN1_REQ_ENGINE_SPEED;
                    Msg.Data[7]           = J1939_PGN2_REQ_ENGINE_SPEED;
                }
                else
                {
                    Msg.Priority           = J1939_INFO_PRIORITY;
                    Msg.DataPage           = J1939_PGN2_REQ_ENGINE_SPEED & 0x01;
                    Msg.PDUFormat          = J1939_PGN1_REQ_ENGINE_SPEED;
                    Msg.GroupExtension     = J1939_PGN0_REQ_ENGINE_SPEED;
                    Msg.DataLength         = 1;
                    Msg.Data[0]           = EngineSpeed;
                }
                while (J1939_EnqueueMessage( &Msg ) != RC_SUCCESS);
            }
        }
    }
}
```


EXAMPLE 5B:

```

/*****
/*****

```

```

/*

```

Example 5b

This example shows what the corresponding node to Example 5A should look like. When the switch is pressed, it will ask Node 0 for the engine speed. If it receives the engine speed, it will display that value on the port D LEDs, matching Node 0's LEDs. If it receives a NACK, it will light the RC2 LED instead.

Application Maestro should be run with the following options changed from their default values (in addition to NAME, Address, and bit rate values):

```

Receive Interrupt Priority - High
Transmit Interrupt Priority - High
*/

```

```

#include <p18cxxx.h>
#include "j1939.h"

```

```

#define NODE1ADDRESS      128
#define NODE2ADDRESS      129

```

```

#define J1939_PGN0_REQ_ENGINE_SPEED      0x04
#define J1939_PGN1_REQ_ENGINE_SPEED      0xf0
#define J1939_PGN2_REQ_ENGINE_SPEED      0x00

```

```

//*****

```

```

void InterruptHandlerHigh (void);

```

```

//-----
// High priority interrupt vector

```

```

#pragma code InterruptVectorHigh = 0x08
void InterruptVectorHigh( void )
{
    _asm
        goto InterruptHandlerHigh
    _endasm
}

```

```

//-----
// High priority interrupt routine

```

```

#pragma code
#pragma interrupt InterruptHandlerHigh

```

```

void InterruptHandlerHigh( void )
{
    if (PIR3 != 0x00)
        J1939_ISR();
}

```

```

//*****

```

```

J1939_MESSAGE      Msg;

```

```

//*****
void main()
{

```

```
unsigned char    LastSwitchRB4 = 1;
unsigned char    CurrentSwitch;
unsigned char    i;

RCONbits.IPEN = 1;
TRISBbits.TRISB4 = 1;
TRISBbits.TRISB5 = 1;
TRISCbits.TRISC2 = 0;
TRISD = 0;

LATD = 0;
LATCbits.LATC2 = 0;

J1939_Initialization( TRUE );

INTCONbits.GIEH = 1;

// Wait for address contention to time out
while (J1939_Flags.WaitingForAddressClaimContention)
    J1939_Poll(5);

// Now we know our address is good, so start checking for
// messages and switches.

while (1)
{
    CurrentSwitch = PORTBbits.RB4;
    if (CurrentSwitch && !LastSwitchRB4)
    {
        // Ask for the engine speed
        Msg.DataPage          = 0;
        Msg.PDUFormat         = J1939_PF_REQUEST;
        Msg.Priority          = J1939_CONTROL_PRIORITY;
        Msg.DestinationAddress = NODE1ADDRESS;
        Msg.DataLength        = 3;
        Msg.Data[0] = J1939_PGN0_REQ_ENGINE_SPEED;
        Msg.Data[1] = J1939_PGN1_REQ_ENGINE_SPEED;
        Msg.Data[2] = J1939_PGN2_REQ_ENGINE_SPEED;
        while (J1939_EnqueueMessage( &Msg ) != RC_SUCCESS);
    }
    LastSwitchRB4 = CurrentSwitch;

    while (RXQueueCount > 0)
    {
        J1939_DequeueMessage( &Msg );
        if (Msg.PDUFormat == J1939_PF_ACKNOWLEDGMENT)
        {
            LATD = 0;
            LATCbits.LATC2 = 1;
        }
        else if ((Msg.PDUFormat == J1939_PGN1_REQ_ENGINE_SPEED) &&
            (Msg.GroupExtension == J1939_PGN0_REQ_ENGINE_SPEED))
        {
            LATD = Msg.Data[0];
            LATCbits.LATC2 = 0;
        }
    }
}
}
```

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Information contained in this publication regarding device applications and the like is intended through suggestion only and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. No representation or warranty is given and no liability is assumed by Microchip Technology Incorporated with respect to the accuracy or use of such information, or infringement of patents or other intellectual property rights arising from such use or otherwise. Use of Microchip's products as critical components in life support systems is not authorized except with express written approval by Microchip. No licenses are conveyed, implicitly or otherwise, under any intellectual property rights.

Trademarks

The Microchip name and logo, the Microchip logo, Accuron, dsPIC, KEELoQ, microID, MPLAB, PIC, PICmicro, PICSTART, PRO MATE, PowerSmart, rfPIC, and SmartShunt are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.


AmpLab, FilterLab, MXDEV, MXLAB, PICMASTER, SEEVAL, SmartSensor and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Analog-for-the-Digital Age, Application Maestro, dsPICDEM, dsPICDEM.net, dsPICworks, ECAN, ECONOMONITOR, FanSense, FlexROM, fuzzyLAB, In-Circuit Serial Programming, ICSP, ICEPIC, Migratable Memory, MPASM, MPLIB, MPLINK, MPSIM, PICkit, PICDEM, PICDEM.net, PICLAB, PICtail, PowerCal, PowerInfo, PowerMate, PowerTool, rLAB, rfPICDEM, Select Mode, Smart Serial, SmartTel and Total Endurance are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.

© 2004, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

 Printed on recycled paper.

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
== ISO/TS 16949:2002 ==

Microchip received ISO/TS-16949:2002 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona and Mountain View, California in October 2003. The Company's quality system processes and procedures are for its PICmicro® 8-bit MCUs, KEELoQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

Worldwide Sales and Service

AMERICAS

Corporate Office

2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support: 480-792-7627
Web Address: www.microchip.com

Atlanta

3780 Mansell Road, Suite 130
Alpharetta, GA 30022
Tel: 770-640-0034
Fax: 770-640-0307

Boston

2 Lan Drive, Suite 120
Westford, MA 01886
Tel: 978-692-3848
Fax: 978-692-3821

Chicago

333 Pierce Road, Suite 180
Itasca, IL 60143
Tel: 630-285-0071
Fax: 630-285-0075

Dallas

4570 Westgrove Drive, Suite 160
Addison, TX 75001
Tel: 972-818-7423
Fax: 972-818-2924

Detroit

Tri-Atria Office Building
32255 Northwestern Highway, Suite 190
Farmington Hills, MI 48334
Tel: 248-538-2250
Fax: 248-538-2260

Kokomo

2767 S. Albright Road
Kokomo, IN 46902
Tel: 765-864-8360
Fax: 765-864-8387

Los Angeles

18201 Von Karman, Suite 1090
Irvine, CA 92612
Tel: 949-263-1888
Fax: 949-263-1338

San Jose

1300 Terra Bella Avenue
Mountain View, CA 94043
Tel: 650-215-1444
Fax: 650-961-0286

Toronto

6285 Northam Drive, Suite 108
Mississauga, Ontario L4V 1X5, Canada
Tel: 905-673-0699
Fax: 905-673-6509

ASIA/PACIFIC

Australia

Suite 22, 41 Rawson Street
Epping 2121, NSW
Australia
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing

Unit 706B
Wan Tai Bei Hai Bldg.
No. 6 Chaoyangmen Bei Str.
Beijing, 100027, China
Tel: 86-10-85282100
Fax: 86-10-85282104

China - Chengdu

Rm. 2401-2402, 24th Floor,
Ming Xing Financial Tower
No. 88 TIDU Street
Chengdu 610016, China
Tel: 86-28-86766200
Fax: 86-28-86766599

China - Fuzhou

Unit 28F, World Trade Plaza
No. 71 Wusi Road
Fuzhou 350001, China
Tel: 86-591-7503506
Fax: 86-591-7503521

China - Hong Kong SAR

Unit 901-6, Tower 2, Metroplaza
223 Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852-2401-1200
Fax: 852-2401-3431

China - Shanghai

Room 701, Bldg. B
Far East International Plaza
No. 317 Xian Xia Road
Shanghai, 200051
Tel: 86-21-6275-5700
Fax: 86-21-6275-5060

China - Shenzhen

Rm. 1812, 18/F, Building A, United Plaza
No. 5022 Binhe Road, Futian District
Shenzhen 518033, China
Tel: 86-755-82901380
Fax: 86-755-8295-1393

China - Shunde

Room 401, Hongjian Building, No. 2
Fengxiangnan Road, Ronggui Town, Shunde
District, Foshan City, Guangdong 528303, China
Tel: 86-757-28395507 Fax: 86-757-28395571

China - Qingdao

Rm. B505A, Fullhope Plaza,
No. 12 Hong Kong Central Rd.
Qingdao 266071, China
Tel: 86-532-5027355 Fax: 86-532-5027205

India

Divyasree Chambers
1 Floor, Wing A (A3/A4)
No. 11, O'Shaughnessy Road
Bangalore, 560 025, India
Tel: 91-80-22290061 Fax: 91-80-22290062

Japan

Benex S-1 6F
3-18-20, Shinyokohama
Kohoku-Ku, Yokohama-shi
Kanagawa, 222-0033, Japan
Tel: 81-45-471-6166 Fax: 81-45-471-6122

Korea

168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku
Seoul, Korea 135-882
Tel: 82-2-554-7200 Fax: 82-2-558-5932 or
82-2-558-5934

Singapore

200 Middle Road
#07-02 Prime Centre
Singapore, 188980
Tel: 65-6334-8870 Fax: 65-6334-8850

Taiwan

Kaohsiung Branch
30F - 1 No. 8
Min Chuan 2nd Road
Kaohsiung 806, Taiwan
Tel: 886-7-536-4818
Fax: 886-7-536-4803

Taiwan

Taiwan Branch
11F-3, No. 207
Tung Hua North Road
Taipei, 105, Taiwan
Tel: 886-2-2717-7175 Fax: 886-2-2545-0139

EUROPE

Austria

Durisolstrasse 2
A-4600 Wels
Austria
Tel: 43-7242-2244-399
Fax: 43-7242-2244-393

Denmark

Regus Business Centre
Lautrup høj 1-3
Ballerup DK-2750 Denmark
Tel: 45-4420-9895 Fax: 45-4420-9910

France

Parc d'Activite du Moulin de Massy
43 Rue du Saule Trapu
Batiment A - 1er Etage
91300 Massy, France
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany

Steinheilstrasse 10
D-85737 Ismaning, Germany
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Italy

Via Quasimodo, 12
20025 Legnano (MI)
Milan, Italy
Tel: 39-0331-742611
Fax: 39-0331-466781

Netherlands

Waagenburghtplein 4
NL-5152 JR, Drunen, Netherlands
Tel: 31-416-690399
Fax: 31-416-690340

United Kingdom

505 Eskdale Road
Winnersh Triangle
Wokingham
Berkshire, England RG41 5TU
Tel: 44-118-921-5869
Fax: 44-118-921-5820