J1939 C Library for PIC16 Microcontrollers and MCP2515 User's Guide

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.

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.

Table of Contents

NOTES:

Preface

HIGHLIGHTS

This section contains general information that will be useful to know before using the J1939 C Library for PIC16 Microcontrollers and MCP2515 User's Guide.

The topics discussed in this section are:

- About This Guide
- Recommended Reading
- The Microchip Internet Web Site
- Customer Support

ABOUT THIS GUIDE

Document Layout

- **Chapter 1: Introduction –** provides an overview of the libraries and identifies their level of support and limitations.
- **Chapter 2: How to use the Library –** lists the source files and how/where they are included in the user's source files and directory. Also describes the basic setup when using interrupt or polling methods and defines the J1939 message structure.
- **Chapter 3: Library Configuration –** describes the necessary steps to properly configure the libraries.
- **Chapter 4: Library Functions –** provides a detailed description of the the library functions.
- **Appendix A: Examples –** provides some example source code to help the user get started using the libraries.
- **Worldwide Sales and Service –** gives the address, telephone and fax number for Microchip Technology Inc. sales and service locations throughout the world.

RECOMMENDED READING

The following is recommended reading.

MCP2515 Data Sheet (DS21801)

This document is available on Microchip's web site at www.microchip.com.

THE MICROCHIP INTERNET WEB SITE

Microchip provides easy access to our documentation and on-line support through our World Wide Web Site at www.microchip.com. You can download files from the web site or from our FTP site at ftp://ftp.microchip.com.

CUSTOMER SUPPORT

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Field Application Engineer (FAE)
- Corporate Applications Engineer (CAE)
- Hot Line

Customers should call their distributor, representative or field application engineer (FAE) for support. Local sales offices are also available to help customers. See the back cover for a listing of sales offices and locations.

Corporate applications engineers (CAEs) may be contacted at (480) 786-7627.

In addition, there is a Systems Information and Upgrade Line. This line provides system users a listing of the latest versions of all of Microchip's development systems software products. Plus, this line provides information on how customers can receive any currently available upgrade kits.

The Hot Line Numbers are:

- 1-800-755-2345 for U.S. and most of Canada, and
- 1-480-786-7302 for the rest of the world.

Chapter 1. Introduction

1.1 OVERVIEW

The J1939 C Library is targeted for use with PIC16 microcontroller applications written with HI-TECH's PICC™ C compiler using the MCP2515 Stand-alone CAN Controller. It offers J1939 communications protocol support for single address capable Controller Applications (CA's) that are either non-configurable, service configurable or command configurable. The library routines handle initialization and network functions automatically. User 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.

1.2 J1939 SUPPORT AND LIMITATIONS

The following address capabilities are supported:

- Non-configurable Address CA's
- Service Configurable CA's
- Command Configurable CA's

The following address capabilities are NOT supported:

- Self-configurable CA's
- Arbitrary Address Capable CA's

Both PDU1 and PDU2 formats are supported.

Broadcast Announce Message format is 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 ignored.

Working Sets are not supported.

Refer to SAE specifications J1939, J1939-21 and J1939-81 for more information on the J1939 specification.

NOTES:

Chapter 2. How to Use the Library

2.1 INTRODUCTION

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 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.

2.1.1 Project files

The following C files must be compiled and linked with the main CA file(s):

- `j1939_16.c`
- `spi16.c`

The following header file must be included into any CA C files that utilize the library routines or any J1939 definitions:

- `j1939cfg.h`

The file, **`j1939cfg.h`**, also includes the following files. These files do not have to be explicitly included into the CA C files:

- `spi16.h`
- `mcp2515.h`
- `j1939_16.h`
- `j1939pro.h`

All of these files must be located in the same directory. They do not have to be in the project directory, but since **`j1939cfg.h`** will change for each CA (the NAME, Address, etc. will be different), each project will need its own unique copy of the files. You may want to copy them into a sub-directory under the main project directory. Do NOT simply rename **`j1939cfg.h`**, since other library files use it.

2.2 BASIC SETUP

The library can be configured to use either the MCP2515's interrupt capability or to poll the device.

2.2.1 Using Interrupts

Interrupts are the preferred method of operation, since it decreases the likelihood that received messages will be missed. Refer to **3.1.1 "Hardware Configuration"** for information on what pins to connect to support interrupts.

There are three functions that need to be called for basic J1939 function support (refer to **Chapter 4. "Library Functions"** for the details of these functions):

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

After performing any self-test or other initialization, call **`J1939_Initialization`** to setup the J1939 support and begin the process of establishing the CA's presence on the network. 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 less than 128. After **`J1939_Initialization`** is called, global interrupts can be enabled.

After initialization, call **`J1939_Poll`** 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, the system is on the network with an established address. If this flag is set, the CA does not have an accepted address on the network.

If the CA cannot accept the Commanded Address message, then it is not necessary to call **`J1939_Poll`** after **`WaitingForAddressClaimContention`** is clear. However, if the Commanded Address message can be accepted, then call **`J1939_Poll`** 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.

The CA's interrupt handler must be declared in the following manner:

```
#pragma interrupt_level 0
void interrupt MyISR( void )
```

This clarifies to the compiler that functions called by **`J1939_Initialization`** will not be called simultaneously during the interrupt handling. Because of this, it is vital that global interrupts not be enabled until after **`J1939_Initialization`** is called.

The CA's interrupt handler must call **`J1939_ISR`** if the INTF flag is 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 INTF flag.

2.2.2 Using Polling

If necessary, the MCP2515 can be polled to transmit and receive messages. Polling may be required if the external interrupt cannot be used due to hardware design constraints. If using polling, ensure that the MCP2515 is polled often enough to avoid missing a received message.

There are two functions that need to be called for basic J1939 function support (refer to **Chapter 4. "Library Functions"** for the details of these functions):

- `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. 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 less than 128.

After initialization, call **`J1939_Poll`** 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, the system is on the network with an established address. If this flag is set, the CA does not have an accepted address on the network.

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.

2.3 MESSAGES

2.3.1 Message Definition and Structure

Create one or more message buffers using the following definition:

```
J1939_USER_MSG_BANK 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_16.h`**, but here are the main fields that the CA will use. Refer to the J1939 Specification for details on each portion of the message.

- `MyMessage.Msg.DataPage`, one bit
- `MyMessage.Msg.Priority`, three bits
- `MyMessage.Msg.PDUFormat`, one byte
- `MyMessage.Msg.PDUSpecific`, one byte. This field can also be referenced as `DestinationAddress` or `GroupExtension` to help clarify the CA code.
- `MyMessage.Msg.SourceAddress`, one byte (this is automatically filled in by the library before transmission)
- `MyMessage.Msg.DataLength`, 4 bits, but must be between 0 and 8
- `MyMessage.Msg.Data[8]`, array of 8 bytes

2.3.2 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

Call the routine **`J1939_DequeueMessage`** to pull one message out of the receive queue and place it in a buffer for processing. Check the variable **`RXQueueCount`** to see if there are any messages ready in the queue. Check the flag **`J1939_Flags.Flags.ReceivedMessagesDropped`** to see if any messages have been dropped. Refer to the **4.2.1 "unsigned char J1939_DequeueMessage (J1939_USER_MSG_BANK J1939_MESSAGE *MsgPtr);"** for more details.

2.3.3 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 **4.2.2 "unsigned char J1939_EnqueueMessage (J1939_USER_MSG_BANK J1939_MESSAGE *MsgPtr);"** 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`**.

2.3.4 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.

2.3.5 Using the Commanded Address Message

If the library has been configured to allow the Commanded Address message, 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, the CA will no longer be able to transmit messages. It 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, per the J1939-21 specification. Refer to **Appendix A. "Examples"**, Example A-3, for Commanded Address transmission example.

NOTES:

Chapter 3. Library Configuration

3.1 INTRODUCTION

The library is configured through a single C header file, **`j1939cfg.h`**. Additionally, one C function may be required in the CA code.

3.1.1 Hardware Configuration

3.1.1.1 NON-CONFIGURABLE ITEMS

Some hardware aspects cannot be configured, since they would make the library more difficult to use. These items are as follows:

- If interrupts are used, the MCP2515 INT pin must be connected to the PIC® device's INT pin (RB0)
- The library does not use the nRXnBF and nTXnRTS pins

3.1.1.2 INTERRUPTS

If interrupts are used, connect the MCP2515 INT pin to the PIC device's INT pin (RB0).

3.1.1.3 MCP2515 CHIP SELECT PIN

Configure the MCP2515 chip select pin to any allowable PIC device's output pin by setting **`J1939_CS_PIN`** and **`J1939_CS_TRIS`** to a valid pin and TRIS pair, per the HI-TECH PICC header file.

3.1.1.4 SPI™ MODE

Set the SPI mode to either of the allowable MCP2515 SPI modes by setting **`J1939_SPI_MODE`** to either **`MODE_00`** for `0,0` or **`MODE_11`** for `1,1`.

3.1.1.5 SPI SPEED

Set the SPI speed to FOSC/4, FOSC/16 or FOSC/64 by setting **`J1939_SPI_SPEED`** to **`SPI_FOSC_4`**, **`SPI_FOSC_16`** or **`SPI_FOSC_64`**.

3.1.1.6 SPI DATA INPUT SAMPLE PHASE

Set the SPI data input sample phase to sample at either the end or the middle of the data output time by setting **`J1939_SAMPLE`** to either **`SMPEND`** or **`SMPMID`**.

3.1.1.7 CLKOUT/SOF

The library does not use the CLKOUT/SOF pin. To enable the CLKOUT pin, set **`J1939_CLKOUT`** to **`CLKOUT_ENABLE`** and set **`J1939_CLKOUT_PS`** to **`CLKOUT_PS1`**, **`CLKOUT_PS2`**, **`CLKOUT_PS4`** or **`CLKOUT_PS8`** for prescalers of FCLOCKOUT of System Clock/1, 2, 4 or 8. Otherwise, set **`J1939_CLKOUT`** to **`CLKOUT_DISABLE`**. If the CA uses this feature, ensure that other MCP2515 functions are not altered.

3.1.1.8 CAN BIT TIMING

The CAN bit timing is set by the values of **J1939_CNF1**, **J1939_CNF2** and **J1939_CNF3**. Refer to the MCP2515 Data Sheet (DS21801), **Section 5.0 "Bit Timing**", for details on defining these values. Take care that all nodes in the system have identical bit timing.

3.1.2 Software Configuration

Many items of the software can be configured. These items are set up in the **j1939cfg.h** header file and are broken out as follows:

3.1.2.1 INTERRUPTS VS. POLLING

If interrupts are used to detect when messages are ready to be received or transmitted, **J1939_POLL_MCP** must not be defined. If polling is used, then **J1939_POLL_MCP** must be defined.

3.1.2.2 J1939 CONFIGURATION

- If the CA can accept the Commanded Address message, define the label **J1939_ACCEPT_CMDADD**. It is important that this label is defined 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 '1' if the Commanded Address message can be accepted and a '0' if it must be ignored. In the case of a service configurable CA, the function may check 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 '1'. The function must have the prototype:

  ```
  unsigned char CA_AcceptCommandedAddress( void );
  ```

- Define the initial CA Address value by setting **J1939_STARTING_ADDRESS** to a valid initial address. Note that this address value must be a valid value as per the J1939 specification.
- Set the 8-byte CA NAME by defining **J1939_CA_NAMEx** to the individual NAME bytes, where 'x' goes from 0 to 7 and 0 is the Least Significant Byte of the NAME.

  For example, suppose a CA's NAME fields have the following values (refer to the SAE J1939 Specification for the permissible values for each field):

Arbitrary Address Capable (1 bit)	0	Not capable
Industry Group (3 bits)	3	Construction equipment
Vehicle System Instance (4 bits)	0	First instance
Vehicle System (7 bits)	0	Non-specific system
(Reserved) (1 bit)	0	
Function (8 bits)	0x81	Laser receiver
Function Instance (5 bits)	0	First instance
ECU Instance (3 bits)	0	First instance
Manufacturer Code (11 bits)	8	Caterpillar, Inc.
Identity Number (21 bits)	0x16D35A	Unique for the application

The complete NAME value would then be: 0x300081000116D35A. Note that the three Least Significant bits of the manufacturer code are mapped into the three Most Significant bits of the Most Significant Byte of the identity number. This NAME value should be mapped into the following #defines:

```
#define J1939_CA_NAME7      0x30
#define J1939_CA_NAME6      0x00
#define J1939_CA_NAME5      0x81
#define J1939_CA_NAME4      0x00
#define J1939_CA_NAME3      0x01
#define J1939_CA_NAME2      0x16
#define J1939_CA_NAME1      0xD3
#define J1939_CA_NAME0      0x5A
```

3.1.2.3 MESSAGE QUEUE CONFIGURATION

- Define the size of the received message queue by setting the value of **J1939_RX_QUEUE_SIZE**. This value must be greater than or equal to '1'. Refer to **Section 3.1.2.4 "RAM Management"** for more information on setting up the queues.
- Define the size of the transmit message queue by setting the value of **J1939_TX_QUEUE_SIZE**. This value must be greater than or equal to '1'. Refer to **Section 3.1.2.4 "RAM Management"** for more information on setting up the queues.
- If the receive queue is full and another message is received, this message can either be dropped or it can overwrite the previous message. If **J1939_OVERWRITE_RX_QUEUE** is defined as **J1939_TRUE**, the new message overwrites the previous message. If it is defined as **J1939_FALSE**, the message is dropped and a flag is set to indicate that received messages have been dropped.
- If the transmit queue is full and another message is queued for transmit, this message can either be dropped or it can overwrite the previous message. If **J1939_OVERWRITE_TX_QUEUE** is defined as **J1939_TRUE**, the new message overwrites the previous message. If it is defined as **J1939_FALSE**, the message is dropped and a return code indicates that the message was not queued.

3.1.2.4 RAM MANAGEMENT

- For optimal RAM allocation, the queues can be moved into different banks. Set **J1939_RX_QUEUE_BANK** and/or **J1939_TX_QUEUE_BANK** to Bank 1, Bank 2 or Bank 3 (dependent on the device) as necessary to balance the queues and CA RAM usage.
- Due to HI-TECH PICC RAM addressing constraints, the library must also know the bank where the message buffers that are passed into **J1939_EnqueueMessage** and **J1939_DequeueMessage** are located. This bank does not have to match either **J1939_RX_QUEUE_BANK** or **J1939_TX_QUEUE_BANK.** Set **J1939_USER_MSG_BANK** to Bank 1, Bank 2 or Bank 3 (dependent on the device) and define any message structures used by the CA as follows:

```
J1939_USER_MSG_BANK J1939_MESSAGE MyMessage;
```

Note the following when deciding which bank to place the queues and the size of the queues:

- Each message in the queue requires 13 bytes of RAM.
- The bank containing the receive queue will have an additional 16 bytes used by the library (8 bytes if the Commanded Address message is not allowed).
- The bank containing the transmit queue will have an additional 13 bytes used by the library.

3.1.2.5 STACK VS. ROM TRADE-OFFS

By default, the library is configured for optimal ROM usage. In this configuration, the following limitations apply:

- Interrupt processing will require five stack levels, which includes the level used by the interrupt itself. This limits the mainline application to three levels. If function calls are nested more than three levels, it puts the application at risk for a stack overflow unless interrupts are disabled during those times.
- **`J1939_Poll`** requires three stack levels, including the level needed to call the function itself. Therefore, **`J1939_Poll`** may be called only from the mainline code, if using interrupts.
- **`J1939_EnqueueMessage`** and **`J1939_DequeueMessage`** require two stack levels, including the level needed to call the function itself. Therefore, these routines may be called either from the mainline or from another function called from the mainline, if using interrupts.

If the CA has ROM to spare and requires more stack space, the library can be configured to use less stack space at the expense of using more ROM. Uncomment the line:

```
#define SPI_USE_ONLY_INLINE_DEFINITIONS
```

in **`j1939cfg.h`** to get the following limitations:

- Interrupt processing will require four stack levels, which includes the level used by the interrupt itself. This limits the mainline application to four levels. If function calls are nested more than four levels, it puts the application at risk for a stack overflow unless interrupts are disabled during those times.
- **`J1939_Poll, J1939_EnqueueMessage,`** and **`J1939_DequeueMessage`** require two stack levels, including the level needed to call the function itself. Therefore, these routines may be called from either the mainline, or from another function up to two levels deep from the mainline, if using interrupts.

Chapter 4. Library Functions

4.1 INTRODUCTION

4.1.1 Interface Variables

4.1.1.1 CA ADDRESS

The CA Address can be accessed through the variable **J1939_Address**. Under normal operation, this value should not be required by the CA and the CA MUST NOT modify this variable. Messages that are transmitted by using **J1939_EnqueueMessage** automatically have this value inserted into the Source Address portion of the message.

4.1.1.2 CA NAME

The CA NAME can be accessed (and modified if necessary) through the array **CA_Name**. This is an array of unsigned chars, stored Least Significant Byte to Most Significant Byte.

4.1.1.3 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.Flags.CannotClaimAddress** – set to '1' if either an address has not yet been claimed or the address cannot be claimed.
- **J1939_Flags.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.Flags.ReceivedMessagesDropped** – set if **J1939_OVERWRITE_RX_QUEUE** is defined as **J1939_FALSE** and received messages have been dropped because the queue was full. The CA must clear this flag.

The CA can alter only **J1939_Flags.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.

4.2 EXTERNAL INTERFACE ROUTINES

4.2.1 unsigned char J1939_DequeueMessage (J1939_USER_MSG_BANK J1939_MESSAGE *MsgPtr);

This function pulls a received message out of the queue and places it into the buffer pointed to by **MsgPtr**. The following status values are returned:

- **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.

4.2.2 unsigned char J1939_EnqueueMessage (J1939_USER_MSG_BANK J1939_MESSAGE *MsgPtr);

This function takes the message in the CA's RAM pointed to by **MsgPtr** and places it in the queue for transmission. While the message will automatically have the CA's Source Address placed in the proper field, the CA must fill in the other values. The following status values are returned:

- **RC_SUCCESS** – Message dequeued successfully.
- **RC_QUEUEFULL** – Transmit queue full; message not queued.
- **RC_CANNOTTRANSMIT** – System cannot currently transmit messages.

4.2.3 void J1939_Initialization(void);

This function must be called after any CA self-test and basic initialization. It initializes the library's global variables, the SPI port, the MCP device and interrupts, if necessary. It then initiates the process of establishing the CA's address on the network.

4.2.4 void J1939_ISR(void);

The CA must call this inline function if it receives an interrupt and the INTF flag is set. This function then calls **J1939_ReceiveMessage** to process any received messages and **J1939_TransmitMessage** to transmit any messages in the transmit queue. It then clears the INTF flag. Since this function is implemented inline, it does not use a stack level.

4.2.5 void J1939_Poll(unsigned char ElapsedTime);

After **J1939_Initialization**, this function must be called every few milliseconds until **J1939_Flags.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 every few milliseconds during the CA's functioning to check for received messages. If interrupts are used and the Commanded Address message can be accepted, this function must still be called every few milliseconds during the CA's main processing to check for address contention in response to claiming a Commanded Address. If interrupts are used, **J1939_Poll** will not check for received messages or messages to transmit, but will allow the **J1939_ISR** to handle that processing. **ElapsedTime** can either be a value calculated at run time or be a constant value. Round down to the nearest millisecond to ensure that the minimum 250 ms contention wait time is met.

4.3 INTERNAL ROUTINES

For reference, this is a list of the internal library routines. It should not be necessary for the CA to utilize these functions, but advanced applications may find them useful.

4.3.1 void J1939_AddressClaimHandling(unsigned char Mode);

This routine is called internally when either the CA must claim its address on the bus or another CA is trying to claim the same address on the bus. The CA must either defend itself or relinquish the address.

4.3.2 void J1939_ReceiveMessages(void);

This routine is called internally from either **J1939_ISR** when an interrupt is received from the MCP2515 or from **J1939_Poll**. If messages have been received, they are read in. Network management messages are processed, while other messages are placed in the receive queue for the CA. If interrupts are used, the CA is responsible for checking the INTF flag and calling **J1939_ISR** if it is set. If the receive queue is full and another message has been received, **J1939_Flags.Flags.ReceivedMessagesDropped** is set.

To reduce stack usage, some message handling is done within this routine:

- Commanded Address Handling: This code is compiled only if **J1939_ACCEPT_CMDADD** is defined. Otherwise, it is not compiled into the library. This section of code is executed if the CA can be commanded to change its address and it is being sent a Commanded Address message. Note that this message must be sent using the Broadcast Announce Message (BAM) protocol. The BAM protocol is only supported to the extent needed to accept the Commanded Address message

4.3.3 void J1939_RequestForAddressClaimHandling(void);

This routine is called internally if the CA has received a Request for Address Claim message. If the CA cannot claim an address, the routine sends out a Cannot Claim Address message. Otherwise, the routine sends out an Address Claim message for the CA's address.

4.3.4 unsigned char J1939_TransmitMessages(void);

This routine is called internally from either **J1939_ISR** when an interrupt is received from the MCP2515 or from **J1939_Poll**. It transmits as many messages from the transmit queue as it can. If the system cannot transmit messages, an error code is returned. If interrupts are used, the CA is responsible for checking the INTF flag and calling **J1939_ISR** if it is set. Note that only two of the three MCP2515 transmit buffers are used to ensure that the messages are actually transmitted on the bus in the order that they were queued. The following status values are returned:

- **RC_SUCCESS** – Message was transmitted successfully.
- **RC_CANNOTTRANSMIT** – System cannot transmit messages. Either the CA cannot claim an address or the MCP2515 is busy.
- **RC_QUEUEEMPTY** – Transmit queue was empty.

NOTES:

Appendix A. Examples

A.1 INTRODUCTION

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 to the next are bolded.

EXAMPLE A-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.

j1939cfg.h should be configured as follows:

J1939_ACCEPT_CMDADD should be commented out
J1939_POLL_MCP should be uncommented
*/

#include <pic.h>
#include ".\j1939\j1939cfg.h"

J1939_USER_MSG_BANK J1939_MESSAGE Msg;

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

   TRISD0 = 1;    // Switch pin
   TRISD1 = 0;    // LED pin
   RD1 = 0;       // Turn off LED
```

EXAMPLE A-1: (CONTINUED)

```
        J1939_Initialization();

        // Wait for address contention to time out
        while (J1939_Flags.Flags.WaitingForAddressClaimContention)
        {
           DelayMilliseconds(1);
           J1939_Poll(1);
        }

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

        while (1)
        {
           CurrentSwitch = RD0;
           if (LastSwitch != CurrentSwitch)
           {
              Msg.Msg.DataPage           = 0;
              Msg.Msg.Priority           = 7;
              Msg.Msg.DestinationAddress    = OTHER_NODE;
              Msg.Msg.DataLength         = 0;
              if (CurrentSwitch == 0)
                 Msg.Msg.PDUFormat = TURN_ON_LED;
              else
                 Msg.Msg.PDUFormat = TURN_OFF_LED;
           while (J1939_EnqueueMessage( &Msg ) != RC_SUCCESS);
              LastSwitch = CurrentSwitch;
           }

           while (RXQueueCount > 0)
           {
              J1939_DequeueMessage( &Msg );
              if (Msg.Msg.PDUFormat == TURN_ON_LED)
                 RD1 = 1;
              else if (Msg.Msg.PDUFormat == TURN_OFF_LED)
                 RD1 = 0;
           }

           // Since we don't accept the Commanded Address message,
           // the value passed here doesn't matter.
           J1939_Poll(1);
        }
     }

     /*************************************************************************/
     /*************************************************************************/
```

EXAMPLE A-2:

```
/*
Example 2

This example shows the same concept as Example 1, except that instead
of polling, it uses interrupts to check for a message to light an LED
and to send a message if a button is pressed.

j1939cfg.h should be configured as follows:

J1939_ACCEPT_CMDADD should be commented out
J1939_POLL_MCP should be commented out
*/

#include <pic.h>
#include ".\j1939\j1939cfg.h"

J1939_USER_MSG_BANK J1939_MESSAGE Msg;

#pragma interrupt_level 0
void interrupt isr( void )
{
   if (INTF)
      J1939_ISR();

}

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

   TRISD0 = 1;    // Switch pin
   TRISD1 = 0;    // LED pin
   RD1 = 0;       // Turn off LED

   J1939_Initialization();
   GIE = 1;

   // Wait for address contention to time out
   while (J1939_Flags.Flags.WaitingForAddressClaimContention)
   {
      DelayMilliseconds(1);
      J1939_Poll(1);
   }

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

EXAMPLE A-2: (CONTINUED)

```
      while (1)
      {
         CurrentSwitch = RD0;
         if (LastSwitch != CurrentSwitch)
         {
            Msg.Msg.DataPage           = 0;
            Msg.Msg.Priority           = 7;
            Msg.Msg.DestinationAddress    = OTHER_NODE;
            Msg.Msg.DataLength         = 0;
            if (CurrentSwitch == 0)
               Msg.Msg.PDUFormat = TURN_ON_LED;
            else
               Msg.Msg.PDUFormat = TURN_OFF_LED;
            while (J1939_EnqueueMessage( &Msg ) != RC_SUCCESS);
            LastSwitch = CurrentSwitch;
         }

         while (RXQueueCount > 0)
         {
            J1939_DequeueMessage( &Msg );
            if (Msg.Msg.PDUFormat == TURN_ON_LED)
               RD1 = 1;
            else if (Msg.Msg.PDUFormat == TURN_OFF_LED)
               RD1 = 0;
         }

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

   /*********************************************************************/
   /*********************************************************************/
```

EXAMPLE A-3:

```
/*
Example 3a

This example shows the same concept as Example 2, using interrupts to
check for a message to light an LED and to send a message if a button
is pressed.  But for the first 5 button presses, the message is sent to
the wrong address.  On the 5th push, the Commanded Address message is
sent to command the other node to use the address that this node is
sending the message to.  Note that this node doesn't even need to know
what the other node's first address is, as long as it knows the node's
NAME.

j1939cfg.h should be configured as follows:

J1939_ACCEPT_CMDADD should be commented out
J1939_POLL_MCP should be commented out
*/

#include <pic.h>
#include ".\j1939\j1939cfg.h"

J1939_USER_MSG_BANK J1939_MESSAGE Msg;

#pragma interrupt_level 0
void interrupt isr( void )
{
   if (INTF)
      J1939_ISR();

}

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

   TRISD0 = 1;    // Switch pin
   TRISD1 = 0;    // LED pin
   RD1 = 0;       // Turn off LED

   J1939_Initialization();
   GIE = 1;
```

EXAMPLE A-3: (CONTINUED)

```
// Wait for address contention to time out
while (J1939_Flags.Flags.WaitingForAddressClaimContention)
{
   DelayMilliseconds(1);
   J1939_Poll(1);
}

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

while (1)
{
   CurrentSwitch = RD0;
   if (LastSwitch != CurrentSwitch)
   {
      Msg.Msg.DataPage           = 0;
      Msg.Msg.Priority           = 7;
      Msg.Msg.DestinationAddress    = SECOND_ADDRESS;
      Msg.Msg.DataLength         = 0;
      if (CurrentSwitch == 0)
         Msg.Msg.PDUFormat = TURN_ON_LED;
      else
      {
         Msg.Msg.PDUFormat = TURN_OFF_LED;
         if (PushCount < 6)
            PushCount ++;
      }
   while (J1939_EnqueueMessage( &Msg ) != RC_SUCCESS);
      LastSwitch = CurrentSwitch;

      if (PushCount == 5)
      {
```

EXAMPLE A-3: (CONTINUED)

```
Msg.Msg.DataPage           = 0;
Msg.Msg.Priority           = 7;
Msg.Msg.DestinationAddress = J1939_GLOBAL_ADDRESS;
Msg.Msg.DataLength         = 8;
Msg.Msg.PDUFormat          = J1939_PF_CM_BAM;
Msg.Msg.Data[0]            = 1939_BAM_CONTROL_BYTE;
Msg.Msg.Data[1]            = 9;    // 9 data bytes
Msg.Msg.Data[2]            = 0;
Msg.Msg.Data[3]            = 2;    // 2 packets
Msg.Msg.Data[4]            = 0xFF; // Reserved
Msg.Msg.Data[5]            = J1939_PGN0_COMMANDED_ADDRESS;
Msg.Msg.Data[6]            = J1939_PGN1_COMMANDED_ADDRESS;
Msg.Msg.Data[7]            = J1939_PGN2_COMMANDED_ADDRESS;
while (J1939_EnqueueMessage( &Msg ) != RC_SUCCESS);

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

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

EXAMPLE A-3: (CONTINUED)

```
            while (RXQueueCount > 0)
            {
               J1939_DequeueMessage( &Msg );
               if (Msg.Msg.PDUFormat == TURN_ON_LED)
                  RD1 = 1;
               else if (Msg.Msg.PDUFormat == TURN_OFF_LED)
                  RD1 = 0;
            }
         }
      }

      /*********************************************************************/
      /*********************************************************************/
```

EXAMPLE A-4:

```
/*
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 (j1939cfg.h)
  - it must have a CA_AcceptCommandedAddress function
  - it must call J1939_Poll during the main loop, even though inter-
    rupts are being used.
The rest of the code is identical.  The change of address will be
handled in the background.

j1939cfg.h should be configured as follows:

J1939_ACCEPT_CMDADD should be uncommented
J1939_POLL_MCP should be commented out
*/

#include <pic.h>
#include ".\j1939\j1939cfg.h"

J1939_USER_MSG_BANK J1939_MESSAGE Msg;

unsigned char CA_AcceptCommandedAddress( void )
{
   return 1;
}

#pragma interrupt_level 0
void interrupt isr( void )
{
   if (INTF)
      J1939_ISR();

}

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

   TRISD0 = 1;    // Switch pin
   TRISD1 = 0;    // LED pin
   RD1 = 0;       // Turn off LED
```

EXAMPLE A-4: (CONTINUED)

```
        J1939_Initialization();
        GIE = 1;

        // Wait for address contention to time out
        while (J1939_Flags.Flags.WaitingForAddressClaimContention)
        {
           DelayMilliseconds(1);
           J1939_Poll(1);
        }

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

        while (1)
        {
           CurrentSwitch = RD0;
           if (LastSwitch != CurrentSwitch)
           {
              Msg.Msg.DataPage           = 0;
              Msg.Msg.Priority           = 7;
              Msg.Msg.DestinationAddress    = OTHER_NODE;
              Msg.Msg.DataLength         = 0;
              if (CurrentSwitch == 0)
                 Msg.Msg.PDUFormat = TURN_ON_LED;
              else
                 Msg.Msg.PDUFormat = TURN_OFF_LED;
              while (J1939_EnqueueMessage( &Msg ) != RC_SUCCESS);
              LastSwitch = CurrentSwitch;
           }

           while (RXQueueCount > 0)
           {
              J1939_DequeueMessage( &Msg );
              if (Msg.Msg.PDUFormat == TURN_ON_LED)
                 RD1 = 1;
              else if (Msg.Msg.PDUFormat == TURN_OFF_LED)
                 RD1 = 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_MS );
        }
     }

     /*********************************************************************/
     /*********************************************************************/
```

EXAMPLE A-5:

```
/*
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.

j1939cfg.h should be configured as follows:

J1939_ACCEPT_CMDADD should be commented out
J1939_POLL_MCP should be commented out
*/

#include <pic.h>
#include ".\j1939\j1939cfg.h"

J1939_USER_MSG_BANK J1939_MESSAGE Msg;

#pragma interrupt_level 0
void interrupt isr( void )
{
   if (INTF)
      J1939_ISR();

}

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

   TRISD0 = 1;    // Switch pin
   TRISD1 = 0;    // LED pin
   RD1 = 0;       // Turn off LED

   J1939_Initialization();
   GIE = 1;

   // Wait for address contention to time out
   while (J1939_Flags.Flags.WaitingForAddressClaimContention)
   {
      DelayMilliseconds(1);
      J1939_Poll(1);
   }

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

EXAMPLE A-5: (CONTINUED)

```
            if (LastSwitch != CurrentSwitch)
            {
               Msg.Msg.DataPage           = 0;
               Msg.Msg.Priority           = 7;
               Msg.Msg.DestinationAddress    = OTHER_NODE;
               Msg.Msg.PDUFormat          = 254;
               Msg.Msg.DataLength         = 0;
               if (CurrentSwitch == 0)
                  Msg.Msg.GroupExtension= TURN_ON_LED;
               else
                  Msg.Msg.GroupExtension= TURN_OFF_LED;
               while (J1939_EnqueueMessage( &Msg ) != RC_SUCCESS);
               LastSwitch = CurrentSwitch;
            }

            while (RXQueueCount > 0)
            {
               J1939_DequeueMessage( &Msg );
               if (Msg.Msg.GroupExtension == TURN_ON_LED)
                  RD1 = 1;
               else if (Msg.Msg.GroupExtension == TURN_OFF_LED)
                  RD1 = 0;
            }
         }
      }

      /*********************************************************************/
      /*********************************************************************/
```

NOTES:

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: http://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'Shaugnessey Road
Bangalore, 560 025, India
Tel: 91-80-2290061 Fax: 91-80-2290062

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 hoj 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 - ler 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
P. A. De Biesbosch 14
NL-5152 SC 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

01/26/04