ssJ1939-Full User's Manual

Created by the J1939 Experts!
Version 1.9 - Revised March 6th, 2013

Simma Software

ssJ1939 Protocol Stack License

READ THE TERMS AND CONDITIONS OF THIS LICENSE AGREEMENT CAREFULLY BEFORE OPENING THE PACKAGE CONTAINING THE PROGRAM DISTRIBUTION MEDIA (DISKETTES, CD, ELECTRONIC MAIL), THE COMPUTER SOFTWARE THEREIN, AND THE ACCOMPANYING USER DOCUMENTATION. THIS SOURCE CODE IS COPYRIGHTED AND LICENSED (NOT SOLD). BY OPENING THE PACKAGE CONTAINING THE SOURCE CODE, YOU ARE ACCEPTING AND AGREEING TO THE TERMS OF THIS LICENSE AGREEMENT. IF YOU ARE NOT WILLING TO BE BOUND BY THE TERMS OF THIS LICENSE AGREEMENT, YOU SHOULD PROMPTLY RETURN THE PACKAGE IN UNOPENED FORM, AND YOU WILL RECEIVE A REFUND OF YOUR MONEY. THIS LICENSE AGREEMENT REPRESENTS THE ENTIRE AGREEMENT CONCERNING THE J1939 PROTOCOL STACK BETWEEN YOU AND SIMMA SOFTWARE, INC. (REFERRED TO AS "LICENSOR"), AND IT SUPERSEDES ANY PRIOR PROPOSAL, REPRESENTATION, OR UNDERSTANDING BETWEEN THE PARTIES.

1. Corporate License Grant. Simma Software hereby grants to the purchaser (herein referred to as the “Client”), a royalty free, non-exclusive license to use the J1939 protocol stack source code (collectively referred to as the "Software”) as part of Client’s product. Except as provided above, Client agrees to not assign, sublicense, transfer, pledge, lease, rent, or share the Software Code under this License Agreement.

2. Simma Software's Rights. Client acknowledges and agrees that the Software and the documentation are proprietary products of Simma Software and are protected under U.S. copyright law. Client further acknowledges and agrees that all right, title, and interest in and to the Software, including associated intellectual property rights, are and shall remain with Simma Software. This License Agreement does not convey to Client an interest in or to the Software, but only a limited right of use revocable in accordance with the terms of this License Agreement.

3. License Fees. The Client in consideration of the licenses granted under this License Agreement will pay a one-time license fee.

4. Term. This License Agreement shall continue until terminated by either party. Client may terminate this License Agreement at any time. Simma Software may terminate this License Agreement only in the event of a material breach by Client of any term hereof, provided that such shall take effect 60 days after receipt of a written notice from Simma Software of such termination and further provided that such written notice allows 60 days for Client to cure such breach and thereby avoid termination. Upon termination of this License Agreement, all rights granted to Client will terminate and revert to Simma Software. Promptly upon termination of this Agreement for any reason or upon discontinuance or abandonment of Client’s possession or use of the Software, Client must return or destroy, as requested by Simma Software, all copies of the Software in Client’s possession, and all other materials pertaining to the Software (including all copies thereof). Client agrees to certify compliance with such restriction upon Simma Software’s request.

5. Limited Warranty. Simma Software warrants, for Client’s benefit alone, for a period of one year (called the “Warranty Period”) from the date of delivery of the software, that during this period the Software shall operate substantially in accordance with the functionality described in the User's Manual. If during the Warranty Period, a defect in the Software appears, Simma Software will make all reasonable efforts to cure the defect, at no cost to the Client. Client agrees that the foregoing constitutes Client’s sole and exclusive remedy for breach by Simma Software of any warranties made under this Agreement. Simma Software is not responsible for obsolescence of the Software that may result from changes in Client’s requirements. The foregoing warranty shall apply only to the most current version of the Software issued from time to time by Simma Software. Simma Software assumes no responsibility for the use of superseded, outdated, or uncorrected versions of the licensed software. EXCEPT FOR THE WARRANTIES SET FORTH ABOVE, THE SOFTWARE, AND THE SOFTWARE CONTAINED THEREIN, ARE LICENSED "AS IS," AND SIMMA SOFTWARE DISCLAIMS ANY AND ALL OTHER WARRANTIES, WHETHER EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

6. Limitation of Liability. Simma Software's cumulative liability to Client or any other party for any loss or damages resulting from any claims, demands, or actions arising out of or relating to this License Agreement shall not exceed the license fee paid to Simma Software for the use of the Software. In no event shall Simma Software be liable for any indirect, incidental, consequential, special, or exemplary damages or lost profits, even if Simma Software has been advised of the possibility of such damages. SOME STATES DO NOT ALLOW THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THE ABOVE LIMITATION OR EXCLUSION MAY NOT APPLY TO CLIENT.

7. Governing Law. This License Agreement shall be construed and governed in accordance with the laws of the State of Indiana.

8. Severability. Should any court of competent jurisdiction declare any term of this License Agreement void or unenforceable, such declaration shall have no effect on the remaining terms hereof.

9. No Waiver. The failure of either party to enforce any rights granted hereunder or to take action against the other party in the event of any breach hereunder shall not be deemed a waiver by that party as to subsequent enforcement of rights or subsequent actions in the event of future breaches.

TABLE OF CONTENTS

Chapter 1

Introduction

ssJ1939 is high performance real-time SAE J1939 protocol stack written in ANSI C. ssJ1939 adheres to both the SAE J1939 specification and to the software development best practices described in the MISRA C guidelines. ssJ1939 has been used in more than 300 different real-time embedded systems and is standard and/or optional equipment on vehicles of every major commercial OEM. ssJ1939 has been certified **J1939 complaint** since 2008, has no reported failures in that time, and passed OEM integration testing with zero errors.

ssJ1939 uses a modularized design with an emphasis on software readability and performance. ssJ1939 is easy to understand and platform independent allowing it to be used on any CPU or DSP with or without an RTOS. ssJ1939 has been shown to be up to 700% faster and 74% smaller than other commercially available J1939 protocol stacks.

ssJ1939 implements the data link layer and transport protocol described in J1939/21 and the network management layer described in J1939/81. The application layer, described in J1939/71, is the responsibility of the end user to implement. Examples of application layer processing are provided in j1939app.c and at the end of this manual.

Filenames	File Description
j1939.c	Core source file for ssJ1939. Do not modify.
j1939.h	Core header file for ssJ1939. Do not modify.
j1939tp.c	Transport protocol source file. Do not modify.
j1939tp.h	Transport protocol header file. Do not modify.
j1939app.c	Application source file for ssJ1939. Modification allowed.
j1939app.h	Application header file for ssJ1939. Modification allowed.
j1939cfg.h	ssJ1939 configuration file. Modification allowed.

Table 1-1: ssJ1939 files

Chapter 2
Integration of ssJ1939

This chapter describes how to integrate ssJ1939 into your system. After this is complete, you will be able to receive and transmit J1939 messages over CAN. For implementation details, please see the chapters covering the APIs for J1939 and CAN.

Integration Steps:

1. Develop or purchase a CAN device driver which adheres to the CAN API specified in Chapter 3.

2. Before using any of the J1939 module features, make sure the CAN driver has been initialized by calling can_init(). Typically it is called shortly after power-on reset and before the application is started.

3. Before using any of the ssJ1939 module features, make sure the ssJ1939 as been initialized by calling j1939_init(). Typically it is called after can_init() and before the application is started.

4. Call j1939_update at a fixed periodic interval (e.g. every 10 ms). This provides the time base for the J1939 module. It is recommended that this function be called at least every 25 ms.

5. Set J1939CFG_TICK_PERIOD, in j1939cfg.h, to your systems fixed periodic interval described above in step #4.

6. Set your “Name field” in j1939cfg.h. See the Configuration chapter for details.

7. As needed adjust the number and size of the transport protocol buffers.

8. As needed place software in the function j1939app_process(), which is located in j1939app.c, to receive and process incoming J1939 messages.

9. As needed, call j1939_tx_app() to transmit any J1939 message, including single and multi-frame messages.

Chapter 3

CAN Hardware Abstraction Layer

The hardware abstraction layer (HAL) is a software module that provides functions for receiving and transmitting controller area network (CAN) data frames. Because CAN peripherals typically differ from one microcontroller to another, this module is responsible for encompassing all platform depended aspects of CAN communications.

The HAL contains three functions that are responsible for initializing the CAN hardware and handling buffered reception and transmission of CAN frames.

Function Prototype	Function Description
void can_init (void)	Initializes CAN hardware
uint8_t can_rx (can_t *frame)	Receives CAN frame (buffered I/O)
uint8_t can_tx (can_t *frame)	Transmits CAN frame (buffered I/O)

Table 3-1: HAL functions

3.1 Data Type Definitions

Data type:

can_t

Description:

can_t is a data type used to store CAN frames. It contains the CAN frame identifier, the CAN frame data, and the size of data. NOTE: If the most significant bit of id (i.e. bit 31) is set, it indicates an extended CAN frame, else it indicates a standard CAN frame.

Definition:

```
typedef struct {

  uint32_t id;
  uint8_t buf[8];
  uint8_t buf_len;

} can_t;
```

3.2 Function APIs

can_init

Function Prototype:

void can_init(void);

Description:

can_init initializes the CAN peripheral for reception and transmission of CAN frames at a network speed of 250 kbps. Any external hardware that needs to be initialized can be done inside of can_init. The sample point should be as close to 0.875 as possible, but should not exceed it. See J1939/11 and J1939/15 for additional bit timing and sample point information.

Parameters:

void

Return Value:

void

can_rx

Function Prototype:

```
uint8_t can_rx ( can_t *frame );
```

Description:

can_rx checks to see if there is a CAN data frame available in the receive buffer. If one is available, it is copied into the can_t structure which is pointed to by frame. If the most significant bit of frame->id (i.e. bit 31) is set, it indicates an extended CAN frame, else it indicates a standard CAN frame.

Parameters:

frame: Points to memory where the received CAN frame should be stored.

Return Value:

1: No CAN frame was read from the receive buffer.
0: A CAN frame was successfully read from the receive buffer.

can_tx

Function Prototype:

```
uint8_t can_tx ( can_t *frame );
```

Description:

If memory is available inside of the transmit buffer, can_tx copies the memory pointed to by frame to the transmit buffer. If transmission of CAN frames is not currently in progress, then it will be initiated. If the most significant bit of frame->id (i.e. bit 31) is set, it indicates an extended CAN frame, else it indicates a standard CAN frame.

Parameters:

frame: Points to the CAN frame that should be copied to the transmit buffer.

Return Value:

1: No CAN frame was written to the transmit buffer.
0: The CAN frame was successfully written to the transmit buffer.

Chapter 4

ssJ1939 Application Program Interface

This chapter describes the application program interface (API) for the J1939 module.

Function Prototypes	Function Descriptions
void j1939_init (void)	Initializes protocol stack
void j1939_update (void)	Provides periodic time base
void j1939app_process (j1939_t *msg)	Processes received J1939 messages
uint8_t j1939_tx_app (j1939_t *msg, uint8_t *status)	Transmits a J1939 message
uint8_t j1939_px (uint16_t,uint16_t,uint8_t(*f)(void))	Periodically transmits J1939 msg
void j1939_bip_tx_rate_allowed_set (uint8_t rate)	Sets maximum transmit rate allowed
uint8_t j1939_bip_tx_rate_max_get (void);	Retrieves peak bus load usage.
uint8_t j1939app_sa_get (void);	Retrieves next source address

Table 4-1: API functions

4.1 Data Type Definitions

Data type:

j1939_t

Description:

j1939_t is a data type used to store J1939 messages. It contains the J1939 message source, destination, PGN, priority, data, and the size of data.

Definition:

```
typedef struct {

  uint32_t pgn;        /* Parameter Group Number. */
  uint8_t *buf;        /* Pointer to data. */
  uint16_t buf_len;    /* Size of data. */
  uint8_t dst;         /* Destination of message. */
  uint8_t src;         /* Source of message. */
  uint8_t pri;         /* Priority of message. */

} j1939_t;
```

j1939_init

Function Prototype:

void j1939_init (void);

Description:

Initializes the J1939 module.

Parameters:

void

Return Value:

void

j1939_update

Function Prototype:

```
void j1939_update ( void );
```

Description:

Provides the periodic time base for the J1939 module.

Parameters:

void

Return Value:

void

j1939app_process

Function Prototype:

```
void j1939app_process ( j1939_t *msg );
```

Description:

Processes received J1939 message. This function is called by the J1939 module with a complete J1939 message and is the intended location for the application layer to handle received J1939 messages. For multipacket messages, this function isn’t called until all packets have been received and assembled into a complete J1939 message.

Parameters:

msg: Pointer to received J1939 message.

Return Value

void

j1939_tx_app

Function Prototype:

```
uint8_t j1939_tx_app ( j1939_t *msg, uint8_t *status );
```

Description:

Buffers a J1939 message for transmission. For messages that are larger than 8 bytes, the dst field of the j1939_t structure controls whether the message is transmitted using the BAM or RTS/CTS procedures. Also, for messages larger than 8 bytes, the application should set status to point to a variable in the application's RAM. *status will be equal to J1939TP_INPROCESS while the message is being transmitted, J1939TP_DONE if the message transmission is complete, or J1939TP_FAILED if there was an error.

Parameters:

msg: Points to the J1939 message that should be transmitted.
status: Points to application RAM. Ignored for messages 8 bytes or less.

Return Value:

1: Message was not buffered for transmission.
0: Message was buffered for transmission.

j1939_px

Function Prototype:

```
uint8_t j1939_px ( uint16_t start, uint16_t period, uint8_t (*f)(void) );
```

Description:

Periodically transmits J1939 messages. This function allows users to request transmission of a message periodically from 0 to 65535 update ticks (see Configuration chapter for more information). For example, if an update rate is configured for a 10 millisecond (ms) period, then this function allows the repetitive transmission period to be between 0 and 65.535 seconds.

The stack statically creates an internal array of function points, period holding variable, and timer. Size of the static array is described in the configuration chapter. Ever time the ssJ1939 updates, it iterates through the array decrementing the timers. When a timer reaches zero, it executes the corresponding function. If the function returns success, the timer is reloaded from the period holding variable, else it is reloaded with a 1 to for another attempt next stack update.

NOTE: Applications should periodically transmit messages back to back. For example, if you have 5 messages that are all transmitted every 100 ms, do not call the j1939_px function with any two having the same start time. In other words, all 100 ms messages should have a unique start value which ensure their timers are out of phase.

Parameters:

Start: Initial timer value.
Period: Transmission period.
F: Function pointer.

Return Value:

1: Message was not added to internal array.
0: Message was added to internal array.

Example:

```
void
j1939app_init ( void )
{
        j1939_px( 10, 100, j1939app_tx_pgn_65222 );
        j1939_px( 20, 100, j1939app_tx_pgn_64300 );
        j1939_px( 30, 100, j1939app_tx_pgn_65522 );
}
```

j1939_bip_tx_rate_allowed_set

Function Prototype:

```
void j1939_bip_tx_rate_allowed_set ( uint8_t rate );
```

Description:

In order to implement babbling idiot protection, the J1939 module tracks how many messages are transmitted by the application in a 250 ms window. If the J1939 module detects the application has transmitted more messages than is allowed, it will permanently disable transmission. The application should use this function to set the J1939 module's allowable transmission rate (default of 25%). Set rate to 100% to disable babbling idiot protection.

Parameters:

rate: Max allowed transmission rate. Range from 0 to 100 percent.

Return Value

void

j1939_bip_tx_rate_max_get

Function Prototype:

```
uint8_t j1939_bip_tx_rate_max_get ( void );
```

Description:

In order to implement babbling idiot protection, the J1939 module tracks how many messages are transmitted by the application in a 250 ms window. The application can use this function to retrieve the max bus load that has been imposed on the bus by the application.

Parameters:

void

Return Value

0 to 100 percent

j1939app_tx_request

Function Prototype:

uint8_t j1939app_tx_request (uint32_t pgn, uint8_t dst);

Description:

Example function which transmits a request for a PGN. The dst field determines if a global request or destination specific request for a parameter group number (PGN) is sent.

Parameters:

pgn: PGN to be requested.

dst: Destination address of request. Use 255 to send request to all nodes. In the J1939 specification, address 255 is the defined to be the global address representing all nodes.

Return Value:

1: Message was not buffered for transmission.

0: Message was buffered for transmission.

j1939app_sa_get

Function Prototype:

```
uint8_t j1939app_sa_get ( void );
```

Description:

This function returns the next source address that the protocol stack should attempt to claim. It is called when the protocol stack fails to claim it's default address, as defined by J1939CFG_SA.

Since the protocol stack has no knowledge of what additional addresses it may claim, it calls this routine which is maintained by the application layer. When the application layer has no more addresses to claim, this function should return 255.

Parameters:

void

Return Value

Next source address. 255 indicates no more addresses are available.

Chapter 5 Configuration

This chapter describes all configurable items of the J1939 module. All of these configurations are defined in j1939cfg.h.

J1939 Update Rate

The update rate is a fixed periodic interval in tenths of a milliseconds, see steps #4 and #5 in Chapter 2, at which the j1939_update function is called. Its default value is 100, which represents 10 milliseconds. Minimum value is 1 (0.1 milliseconds) and the recommended maximum is 250 (25 milliseconds).

```
#define J1939CFG_TICK_PERIOD                    100
```

J1939 Source Address

The source address is a 8-bit field and identifies a unique J1939 device on the network. Its default value is set to an experimental ECU, but should be changed to match it's actual function. Possible values are listed in the J1939 parent document.

```
#define J1939CFG_SA              J1939_SA_EXPERIMENTAL_USE
```

Transport Protocol RX Buffer Count

Multipacket J1939 messages are split into multiple CAN frames and buffered inside of the J1939 module. This configuration defines how many incoming multipacket messages can be received simultaneously.

```
#define J1939CFG_TP_RX_BUF_NUM       10
```

Transport Protocol RX Buffer Size

Multipacket J1939 messages are split into multiple CAN frames and buffered inside of the J1939 module. This configuration defines in bytes the largest message which can be received. The maximum buffer size is1,785 bytes.

```
#define J1939CFG_TP_RX_BUF_SIZE       128
```

Transport Protocol TX Buffer Count

Multipacket J1939 messages are split into multiple CAN frames and buffered inside of the J1939 module. This configuration defines how many outgoing multipacket messages can be transmitted simultaneously.

```
#define J1939CFG_TP_TX_BUF_NUM        3
```

Transport Protocol TX Buffer Size

Multipacket J1939 messages are split into multiple CAN frames and buffered inside of the J1939 module. This configuration defines in bytes the largest message which can be transmitted. The maximum buffer size is1,785 bytes.

```
#define J1939CFG_TP_TX_BUF_SIZE       128
```

J1939 Name Field

The J1939 name field is 64 bits long and is intended to uniquely describe all ECUs on a J1939 network (i.e. no two ECUs on a J1939 network may have the same name field). For more information, see J1939/81 section 4.1.

```
#define J1939CFG_N_AAC       0  /* 1-bit Arbitrary Address Capable */
#define J1939CFG_N_IG        0  /* 3-bit Industry Group */
#define J1939CFG_N_VSI       0  /* 4-bit Vehicle System Instance */
#define J1939CFG_N_VS        0  /* 7-bit Vehicle System */
#define J1939CFG_N_F         0  /* 8-bit Function */
#define J1939CFG_N_FI        0  /* 5-bit Function Instance */
#define J1939CFG_N_EI        0  /* 3-bit ECU Instance */
#define J1939CFG_N_MC        0  /* 11-bit Manufacturer Code */
#define J1939CFG_N_IN        0  /* 21-bit Identity Number */
```

Chapter 6
Examples

This chapter gives examples of how to receive/decode J1939 messages, how to transmit a J1939 message, and to transmit requests periodically. Per the J1939 specification all J1939 messages have an associated PGN followed by data. The below examples show how to filter based on the PGN of interest and decode the data.

All application layer PGNs, the parameters contained in the specific PGN, the parameter's size and location (i.e. offset) inside of the message buffer are fully specified and documented inside of SAE J1939-71. For example, J1939-71 specifies the parameter 'Engine Speed' to be transmitted inside of PGN 61444, to be two bytes in size, and to be located in the 2nd and 3rd bytes of the message buffer (i.e. data field). Receiving and decoding of 'Engine Speed' is show below, in the second case of the switch statement.

The below buffer conversion macros, btou8(), btou16(), and btou32() handle converting a buffer to an uint8_t, uint16_t, and to an uint32_t. The main purpose of the macros, instead of explicit C statements, is to reduce the chances of an error and development time.

6.1 Receive and Decode J1939 Messages Example:

```
void
j1939app_process ( j1939_t *msg )
{
    switch( msg->pgn ) {
        /*    -PGN-   -CUSTOMER VARIABLE-              -BUFFER CONVERSION-      */
        case 61443: j1939_accel_position          = btou8(  &msg->buf[1] ); break;
        case 61444: j1939_engine_speed            = btou16( &msg->buf[3] ); break;
        case 65244: j1939_idle_fuel_used          = btou32( &msg->buf[0] ); break;
        case 65248: j1939_vehicle_distance        = btou32( &msg->buf[4] ); break;
        case 65253: j1939_total_engine_hours      = btou32( &msg->buf[0] ); break;
        case 65257: j1939_total_fuel_used         = btou32( &msg->buf[4] ); break;
        case 65262: j1939_coolant_temp            = btou8(  &msg->buf[0] ); break;
        case 65263: j1939_oil_pressure            = btou8(  &msg->buf[3] ); break;
        case 65265: j1939_vehicle_speed           = btou16( &msg->buf[1] ); break;
        case 65266: j1939_avg_fuel_econ           = btou16( &msg->buf[4] ); break;
        case 65276: j1939_fuel_level              = btou8(  &msg->buf[1] ); break;
    }
}
```

6.2 Transmit J1939 Message Example - Single Frame:

```
void
j1939_transmit_example ( void )
{
  j1939_t  msg;
  uint8_t buf[8] = {0,1,2,3,4,5,6,7};

  /* load message */
  msg.pgn = 65215;
  msg.buf = buf;
  msg.buf_len = 8;
  msg.dst = 255;
  msg.pri = 7;

  /* transmit message */
  if( j1939_tx_app(&msg, 0) == 0 )
    printf("Message transmitted\n");
  else
    printf("Message not transmitted\n");
}
```

6.3 Transmit J1939 Message: Multi-frame

```
void
j1939_transmit_example ( void )
{
  j1939_t  msg;
  uint8_t buf[16] = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};

  /* load message */
  msg.pgn = 65215;
  msg.buf = buf;
  msg.buf_len = 16;
  msg.pri = 7;

  /* 255 sends a multi-frame to global (i.e. BAM) message.  Using a nodes address, like
     0 for the engine, specifies a destination specific multi-frame (RTS/CTS) */
  msg.dst = 255;

  /* transmit message */
  if( j1939_tx_app(&msg, 0) == 0 )
    printf("Message transmitted\n");
  else
    printf("Message not transmitted\n");
}
```

6.4 Periodically Request J1939 Parameters Example:

The below example shows how to transmit requests, for PGNs 65253, 65244, and 65257, periodically using the built-in example function j1939app_tx_request(). The requests are sent every 5 seconds with a 1 second spacing. The below routine assumes the protocol stack's update function is called every 1 millisecond.

```
void
j1939app_update ( void )
{
    static uint16_t time = 0;

    /* transmit requests every 5 seconds */
    switch( time++ ) {
         case 1000: j1939app_tx_request( 65253, 255); break;
         case 2000: j1939app_tx_request( 65244, 255); break;
         case 3000: j1939app_tx_request( 65257, 255); break;
         case 5000: time = 0;
    }
}
```