

ssCAN

User's Manual

Version 1.0
Revised February 29th, 2012
Created by the [CAN](#) Experts!



Simma Software

ssCAN Device Driver 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 CAN DEVICE DRIVER 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 CAN device driver 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

1.0	INTRODUCTION	4
2.0	INTEGRATION	5
3.0	ssCAN API	6
4.0	CONFIGURATION	10
5.0	EXAMPLES	11

Chapter 1

Introduction

ssCAN is an extreme performance CAN bus device driver with low interrupt latency and low CPU overhead, written in ANSI C. ssCAN adheres to both the ISO 11898 specification and to the software development best practices described in the MISRA C guidelines.

ssCAN is a modularized design with an emphasis on software readability and performance. ssCAN is easy to understand and target specific allowing high-level CAN protocols to be completely platform independent. ssCAN has been shown to be up to 800% faster and 80% smaller than other commercially available CAN device drivers.

ssCAN implements the data link layer in accordance to ISO 11898-1, -2, -3, and -5.

Filenames	File Description
can.c	Core source file for ssCAN. Do not modify.
can.h	Core header file for ssCAN. Do not modify.

Table 1-1: ssCAN files

Chapter 2

Integration of ssCAN

This chapter describes how to integrate ssCAN into your application. After this is complete, you will be able to receive and transmit CAN data frames over a CAN bus. For implementation details, please see the chapters covering the CAN API.

Integration Steps:

1. Depending on your target, CAN related interrupts may need to be added to your microcontrollers vector table.
2. Configure your CAN baud rate settings by selecting one of the baud rate configurations in `can.h`
3. Depending on your target, enable corresponding CAN_H and CAN_L GPIO pins for CAN operation.
4. Enable clock to CAN peripheral.
5. Before using any of the modules 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.

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 configured in can.h. Any external hardware that needs to be initialized, for example enabling a CAN transceiver, can be done inside of can_init.

High-level CAN protocols such as SAE J1939 and ISO 15765 recommend a sample point as close to 0.875 as possible, but never exceeding it. See J1939/11 and ISO 15765 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

Configuration

This chapter describes all configurable items of the ssCAN module. All of these configurations are defined at the top of can.c or can.h.

Receive Buffer Count

This configuration defines how many incoming CAN data frames can be held in the receive FIFO.

```
#define CAN_RX_BUF_SIZE          10
```

Transmit Buffer Size

This configuration defines how many incoming CAN data frames can be held in the transmit FIFO.

```
#define CAN_TX_BUF_SIZE          6
```

Chapter 5

Examples

This chapter gives examples of how to receive and receive CAN data frames.

5.1 Transmit CAN Frame Example:

```
void
can_app_tx ( void )
{
    can_t frame;

    frame.id = 0x55;
    frame.buf_len = 1;
    frame.buf[0] = 0x7F;

    /* transmit a CAN frame */
    if( can_tx(&frame) == 0 )
        printf("success\n");
    else
        printf("failure\n");
}
```

5.2 Receive CAN Frame Example:

```
void
can_rx_example ( void )
{
    can_t frame;

    /* receive a CAN frame */
    if( can_rx(&frame) == 0 )
        printf("success, CAN ID is %d\n", frame.id);
    else
        printf("failure\n");
}
```