ssXCP User's Manual

Created by the J1939 Experts!
Visit our XCP Protocol Page.
Version 1.0
Revised August 15th, 2014

ssXCP 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 ssXCP 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 ssXCP 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 breach

TABLE OF CONTENTS

Chapter 1: Introduction

ssXCP is a high performance XCPonCAN protocol stack written in ANSI C. It adheres to both the XCP specification and to the software development best practices described in the MISRA C guidelines.

The XCP protocol stack is a modularized design with an emphasis on software readability and performance. It is easy to understand and platform independent allowing it to be used on any CPU or DSP with or without an RTOS.

Filenames	File Description
xcp.h	Core header file. Do not modify.
xcp.c	Core source file. Do not modify.
xcpcan.h	Transport header file. Do not modify.
xcpcan.c	Transport source file. Do not modify.
xcpcfg.c	Configuration file. Modification allowed.
xcpcfg.h	Configuration file. Modification allowed.
xcpapp.h	Application header file. Modification allowed.
xcpapp.c	Application source file. Modification allowed.

Table 1-1: ssXCP files

Chapter 2: Integration of ssXCP

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

Integration Steps:

1. Develop or purchase a CAN device driver that adheres to the CAN API specified in Chapter 3.
2. Implement the required methods: xcpapp_user_cmd, xcpapp_get_seed, and xcp_unlock; as specified in sections 4.10 through 4.12.
3. Configure the #define section in xcpcfg.h as outlined in chapter 5.
4. If desired, configure the DAQs in xcpcfg.c as outlined in chapter 5.
5. Before using any ssXCP features be sure to call can_init() and xcp_init(), in that order, to reset and initialize both the driver and the protocol stack.
6. Call xcp_update at a fixed periodic interval (e.g. every 5 ms). This provides the time base for the XCP stack.

Chapter 3: ssCAN Driver API

The controller area network (CAN) driver application program interface (API) 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 dependent aspects of CAN communications.

The CAN Driver API 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: CAN Driver API 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, otherwise 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

3.2.1 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 or 500 kbps. Any external hardware that needs to be initialized can be done inside of can_init. The sample point should be as close to 0.80 as possible.

Parameters:

void

Return Value:

void

3.2.2 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, otherwise 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.

3.2.3 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, otherwise 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: ssXCP API

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

Function Prototypes	Function Descriptions
void xcp_init (void)	Initializes protocol stack
void xcpapp_init (void)	Called on startup
void xcp_update (void)	Provides periodic time base
void xcpapp_update (void)	Called at periodic tick rate
void xcptrnsp_tx_rsp (uint8_t *msg, uint8_t len)	Transmits a response message
void xcp_error (uint8_t err, uint8_t *opt, uint8_t optlen)	Transmits an error message
void xcp_ev (uint8_t ev, uint8_t *opt, uint8_t optlen)	Transmits an event message
void xcp_serv (uint8_t serv, uint8_t *opt, uint8_t optlen)	Transmits a service message
void xcp_event (uint8_t chn)	Triggers a DAQ event

Table 3-1: API functions

4.1 xcp_init

Function Prototype:

```
void xcp_init (
    void
);
```

Description:
Initializes and resets the XCP module.

Parameters:
void

Return Value:
void

4.2 xcpapp_init

Function Prototype:

```
void xcpapp_init (
    void
);
```

Description:

Initializes the XCP application software. The implementation of this method is left for the user to add any necessary functions to be called when the XCP module is initialized or reset.

Parameters:

void

Return Value:

void

4.3 xcp_update

Function Prototype:

```
void xcp_update (
    void
);
```

Description:
Provides the periodic time base for the XCP module.

Parameters:
void

Return Value:
void

4.4 xcpapp_update

Function Prototype:

```
void xcpapp_update (
    void
);
```

Description:

This method is called at the tick rate of the XCP module. The user should use this method for any tasks which should be performed on tick.

Parameters:

void

Return Value:

void

4.5 xcptrnsp_tx_rsp

Function Prototype:

```
void xcptrnsp_tx_rsp (
    uint8_t *msg,
    uint8_t  len
);
```

Description:

Transmits fully formed XCP responses. These messages will be converted to the proper format before being transferred over the bus. Messages sent with this method should be responses: they will be sent with the response ID.

Parameters:

msg: Pointer to the message to be transmitted
len: Length of message buffer

Return Value:

void

4.6 xcp_error

Function Prototype:

```
void xcp_error (
    uint8_t err,
    uint8_t *opt,
    uint8_t optlen
);
```

Description:

Transmits an error message with the given error code and optional data. `xcp.h` contains all possible XCP error codes.

Parameters:

err: Error code to be transmitted.
*opt: Buffer containing optional data to be transmitted
optlen: Length of optional buffer

Return Value:

void

4.7 xcp_ev

Function Prototype:

```
void xcp_ev (
    uint8_t ev,
    uint8_t *opt,
    uint8_t optlen
);
```

Description:

Transmits an event message with the given event code and optional data. Xcp.h contains all possible XCP event codes.

Parameters:

ev: Event code to be transmitted
*opt: Buffer containing optional data to be transmitted
optlen: Length of optional buffer

Return Value:

void

4.8 xcp_serv

Function Prototype:

```
void xcp_serv (
    uint8_t serv,
    uint8_t *opt,
    uint8_t optlen
);
```

Description:

Transmits a service message with the given service code and optional data. Xcp.h contains all possible XCP service codes.

Parameters:

serv: Service code to be transmitted
*opt: Buffer containing optional data to be transmitted
optlen: Length of optional buffer

Return Value:

void

4.9 xcp_event

Function Prototype:

```
void xcp_event (
   uint8_t chn
);
```

Description:

This method is called to trigger a DAQ transmitting event. When called, all DAQs on the given channel will transmit their DTOs.

Parameters:

chn: Event channel

Return Value:

void

4.10 xcpapp_user_cmd

Function Prototype:

```
void xcpapp_user_cmd (
    uint8_t *data
);
```

Description:

This method is called when a packet which needs to be processed in a user defined manner. If xcp_success is set to 1 when the method returns, the protocol will send a standard positive response. If any other response is desired, xcp_success should be left as zero and xcptrnsp_tx_rsp() should be used to send a positive response message, or xcp_error() used to send a negative response.

Parameters:

data: Buffer containing data received in message

Return Value:

void

4.11 xcpapp_get_seed

Function Prototype:

```
uint8_t xcpapp_get_seed (
    uint8_t resource,
    uint8_t *buf
);
```

Description:

This method loads the seed for the requested resource into the buffer and returns the length. All seeds must be of length less than MAX_CTO – 1.

Parameters:

resource: Requested resource to be unlocked.
*buf: Seed to be returned, packed into a byte array.

Return Value:

uint8_t Length of seed

4.12 xcpapp_unlock

Function Prototype:

```
uint8_t xcpapp_unlock (
    uint8_t *key,
    uint8_t len
);
```

Description:

This method verifies the received key is valid for unlocking the previously specified resource. If the key is valid, the resource to be unlocked should be returned. Otherwise return 0.

Parameters:

func: Sub-function code
sprsp: Suppress response from server
*key: Timing parameter values to be set in the server
klen: Length of key array in bytes

Return Value:

uint8_t If the key is valid return the resource to be unlocked, otherwise return 0.

Chapter 5: Configuration

This chapter describes all configurable items of the ssXCP module. All of these configurations are defined in xcp_cfg.h and xcp_cfg.c. Remember to also configure the other stack layers per the applicable user manuals.

5.1 XCP Features

These configuration options enable/disable the various features of XCP. Currently the PGM and PAG features are not implemented and should not be enabled.

```
#define XCPCFG_CAL                      1
#define XCPCFG_DAQ                      1
#define XCPCFG_STIM                     1
#define XCPCFG_PGM                      0
#define XCPCFG_PAG                      0
```

5.2 Memory Address Granularity

This variable sets the size of the smallest memory block to be accessed by the protocol. All data transfer size arguments are in terms of the address granularity. All addresses are checked to be aligned with the granularity. Options are 1, 2, and 4 for 1, 2 or 4 bytes.

```
#define XCPCFG_ADDR_GRANU               2
```

5.3 Block Transfer Mode

The block transfer feature for messages being transmitted to the slave can be enabled/disabled with XCPCFG_BLCK_TRNSF.

```
#define XCPCFG_BLCK_TRNSF               1
```

5.4 Additional Communication Features

This variable enables/disables additional communication features. It is a bitmask consisting of two bits. Bit 1 toggles interleaved communication, however it is not supported on CAN and should be set to 0. Bit 0 toggles block transfer from the slave to the master. If master block transfer is enabled two other variables must be configured. The maximum number of packets which can be transferred in a block is controlled with XCPCFG_MAX_BS. The minimum delay between packets is measured in update cycles and is set by XCPCFG_MIN_ST.

```
#define XCPCFG_COMM_OPT                     1
#define XCPCFG_MAX_BS                       4
#define XCPCFG_MIN_ST                       5
```

5.5 Queue Size

This variable sets the number of packets which can be queued to be transferred. It should be set to the number of CAN buffers available for transfer.

```
#define XCPFG_QUEUE_SIZE                    4
```

5.6 Driver Version

This variable sets the reported driver version. It should be set to reflect any driver version iteration to ensure compatibility.

```
#define XCPFG_DRVR_VERS                     1
```

5.7 Block Checksum Maximum Size

This variable sets the maximum size of a memory block which can be checked. This value is in units of the address granularity.

```
#define XCPCFG_MAX_BLCK_SIZE                (0x10000/XCPCFG_ADDR_GRANU)
```

5.8 Unused Configuration Variables

The following variables are unused and should be left to their default setting. They exist to maintain compatibility with the XCP standard and may be used in future versions.

```
#define XCPCFG_INTERLEAVED                            0
#define XCPCFG_FREEZE                                 0
#define XCPCFG_MAX_SEGMENT                            0
#define XCPCFG_TS_MODE                                0
#define XCPCFG_TS_TICK                                0
```

5.9 DAQ/STIM Configuration Variables

The following variables are used to configure the DAQ and STIM features.
MAX_DAQ specifies the number of DAQ structures which can contain data.
MAX_ODT specifies the number of ODTs which can be allocated per DAQ.
MAX_ODTENTRY specifies the number of entries which can be allocated per ODT.
MIN_DAQ specifies the index of the first DAQ which can be configured at runtime.
MAX_EVENT specifies the highest event channel.
DTO_PID specifies which addressing scheme should be used for DTO messages. The three options are: 0 for a unique identifier for each ODT, specified in the first byte.
1 for a unique identifier for each DAQ, specified in the second byte and a relative identifier for each ODT specified in the first.
2 for unique CAN IDs for each ODT, which must be configured.

```
#define XCPCFG_MAX_DAQ                                4
#define XCPCFG_MAX_ODT                                2
#define XCPCFG_MAX_ODTENTRY                           7
#define XCPCFG_MIN_DAQ                                0
#define XCPCFG_MAX_EVENT                              5
#define XCPCFG_DTO_PID                                0
```

5.10 DAQ Configuration

The DAQ engine can be configured both at run-time and compile-time. If they are to be set up at compile-time simply replace the zeroes in the correct struct fields in `xcp_cfg.c`. If more or fewer DAQs are needed they can be added or removed from `xcp_cfg.c`. Undefined behavior will occur if the structs are not configured properly.

5.10.1 DAQ Data Type Description

Data type:

odtentry_t

Description:

odtentry_t is a data type used to store ODT entries.

Definition:

```
typedef struct {

  uint8_t bo; /* bit offset */
  uint8_t len; /* length */
  uint8_t ext; /* address extension */
  uint32_t addr; /* address */

} odtentry_t;
```

Data type:

odt_t

Description:

odt_t is a data type used to store ODTs.

Definition:

```
typedef struct {

  odtentry_t odtentry[XCPCFG_MAX_ODTENTRY]; /* ODTEntry array */
  uint8_t num; /* Number of ODTs */

} odt_t;
```

Data type:
daq_t

Description:
daq_t is a data type used to store DAQs.

Definition:

```
typedef struct {

  odt_t *odt[XCPCFG_MAX_ODT]; /* Pointer array to ODTs in DAQ */
  uint16_t event; /* Event which triggers DAQ */
  uint8_t num; /* Number of ODTs */
  uint8_t mode; /* 0 for DAQ, 1 for STIM */
  uint8_t prop; /* Property bit mask */
  uint8_t trans; /* Prescaler, not supported */
  uint8_t priority; /* Transmit priority */

} daq_t;
```

Data type:
alldaq_t

Description:
alldaq_t is a data type used to store pointers to all of the DAQs.

Definition:

```
typedef struct {

  daq_t *daqs[XCPCFG_MAX_DAQ]; /* Pointer to DAQs */
  uint16_t num; /* Number of DAQs */
  uint8_t prop; /* General DAQ properties */
  uint8_t key; /* DAQ Key Byte */

} alldaq_t;
```

Chapter 6: Examples

This chapter gives examples of how to implement xcpapp_user_cmd, xcpapp_get_seed, and xcp_unlock.

6.1 xcpapp_user_cmd Example:

```
void
xcpapp_user_cmd ( uint8_t *data )
{
  uint8_t *rsp;
  uint8_t rsplen;

  switch( data[0] ) {
    /* Return XCP_MTA pointer address */
    case 0x01:
      buf[0] = 0xff;
      buf[1] = (uint8_t) xcp_mta>>24;
      buf[2] = (uint8_t) xcp_mta>>16;
      buf[3] = (uint8_t) xcp_mta>>8;
      buf[4] = (uint8_t) xcp_mta;
      rsplen = 5;
  }
  xcptrnsp_tx_rsp( rsp, rsplen );
}
```

6.2 xcpapp_get_seed Example

```
uint8_t
xcpapp_get_seed ( uint8_t resource, uint8_t *buf )
{
  buf[0] = 'P';
  buf[1] = 'S';
  buf[2] = 'S';
  buf[3] = 'W';
  buf[4] = 'R';
  buf[5] = 'D';
  xcpapp_resource = resource;
  return 6;
}
```

6.3 xcpapp_unlock Example:

```
uint8_t
xcpapp_unlock ( uint8_t *key, uint8_t len )
{
  if( key[0] == 'P' &&
      key[1] == 'S' &&
      key[2] == 'S' &&
      key[3] == 'W' &&
      key[4] == 'R' &&
      key[5] == 'D' &&
      len == 6 ) {
    return xcpapp_resource;
  }
  return 0;
}
```