ssl14229 Server User's Manual

Created by the UDS Experts!
Version 1.0
Revised May 17, 2016

SIMMA SOFTWARE
EMBEDDED EXCELLENCE

ssI14229 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 ssI14229 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 ssI14229 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

ssI14229 is a high performance ISO 14229 (UDS) protocol stack written in ANSI C. It adheres to both the ISO 14229 specification and to the software development best practices described in the MISRA C guidelines.

The ssI14229 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
uds.h	Core header file. Modification discouraged.
uds.c	Core source file. Modification discouraged.
udsapp.h	Application header file. Modification required.
udsapp.c	Application source file. Modification required.

Table 1-1: ssI14229 files

Chapter 2: Integration of ssI14229

This chapter describes how to integrate ssI14229 into your application. After this is complete, you will be able to receive and transmit ISO 14229 messages over CAN. For implementation details, please see the chapters covering the API for ssI14229.

Integration Steps:

1. Purchase a CAN device driver and integrate it in accordance with the integration specifications for ssI15765.

2. Implement the methods in udsapp as outlined in this document and the ISO 14229-1 specification.

3. As needed adjust the number and size of the message buffer.

Chapter 3: ssI14229 API

This chapter describes the application program interface (API) for the ssI14229 module.

Function Prototypes	**Function Descriptions**
void uds_init (void)	Initializes protocol stack
void udsapp_init (void)	Called on startup
void uds_update (void)	Provides periodic time base
void udsapp_update (void)	Called at periodic tick rate
void udsapp_process (i15765_t *msg)	Processes received messages.
uint8_t udsapp_rddataid (uint16_t data_id, uint8_t *data_len, uint8_t *buf)	Read data by identifier
uint8_t udsapp_rdmem (uint32_t addr, uint32_t size, uint8_t *buf)	Read data from memory
uint8_t udsapp_defdid_id (uint16_t data_id, uint16_t src_id, uint8_t pos, uint8_t len)	Define new data_id from existing id
uint8_t udsapp_defdid_addr (uint16_t data_id, uint32_t addr, uint32_t size)	Define new data_id from memory address
uint8_t udsapp_cleardid (uint16_t data_id)	Clear existing data_id
uint8_t udsapp_write_did (uint16_t data_id, uint8_t *data, uint16_t len)	Write data to data_id
uint8_t udsapp_write_mem (uint32_t addr, uint32_t size, uint8_t *data)	Write data to memory address
uint8_t udsapp_cleardtc (uint32_t dtc)	Clear specified DTC
uint8_t udsapp_iocontrol (uint16_t data_id, uint8_t *buf, uint16_t buf_len, uint8_t *rsp, uint16_t *rsp_len)	Modify input/output from ECU
uint8_t udsapp_routine_start (uint16_t rout_id, uint8_t *control_record, uint16_t rec_len, uint8_t *rsp, uint16_t *rsp_len)	Start a predefined routine
uint8_t udsapp_routine_status(uint16_t rout_id, uint8_t *control_record, uint16_t rec_len, uint8_t *rsp, uint16_t *rsp_len)	Report status of a predefined routine
uint8_t udsapp_reqdl(uint32_t addr, uint32_t size, uint8_t fmt)	Request a download start
uint8_t udsapp_requl(uint32_t addr, uint32_t size, uint8_t fmt)	Request an upload start
uint8_t udsapp_transfer_data(uint8_t bsc, uint8_t *buf, uint16_t *buf_len)	Transfer data to/from the tester
uint8_t udsapp_transfer_exit(uint8_t *trpr, uint16_t trpr_len, uint8_t *rsp, uint16_t *rsp_len)	Exit a download/upload session

uint8_t udsapp_contr_dtc (uint8_t subfunc, uint8_t* opt_rec, uint8_t buf_len)	Controls DTC setting
uint8_t udsapp_reset_timing (void)	Reset modified timing parameters
uint8_t udsapp_clear_roe (void)	Clear all defined response on event flows
uint8_t udsapp_reset_security (void)	Reset security to default state
uint8_t udsapp_secure_access (uint8_t sub_func, uint8_t *rec, uint16_t rec_len, uint8_t *rsp, uint16_t *rsp_len)	Exchange seed/key and verify responses for secure access
uint8_t udsapp_session (uint8_t session_type)	Change the active session
uint8_t udsapp_readdtc (uint8_t sub_func, uint8_t *buf, uint16_t buf_len, uint8_t *rsp, uint16_t *rsp_len)	Read a DTC
uint8_t udsapp_comm_cont (uint8_t sub_func, uint8_t type)	Control the communication mode
uint8_t udsapp_rdper_dataid (uint8_t trans_mode, uint8_t *dids, uint8_t did_cnt, uint8_t *rsp, uint16_t *rsp_len)	Setup a periodic read of a data identifier

Table 3-1: API functions

3.1 uds_init

Function Prototype:

```
void uds_init (
    void
);
```

Description:

Initializes the ssI14229 module.

Parameters:

void

Return Value:

void

3.2 udsapp_init

Function Prototype:

```
void udsapp_init (
    void
);
```

Description:

Initializes the ssI14229 software. The implementation of this method is left for the user to add any necessary functions.

Parameters:

void

Return Value:

void

3.3 uds_update

Function Prototype:

```
void uds_update (
    void
);
```

Description:

Provides the periodic time base for the ssl14229 module.

Parameters:

void

Return Value:

void

3.4 udsapp_update

Function Prototype:

```
void udsapp_update (
    void
);
```

Description:

Provides a regularly called method stub for the user to add methods, which need to be called regularly.

Parameters:

void

Return Value:

void

3.5 uds_process

Function Prototype:

```
void uds_process (
   i15765_t *msg
);
```

Description:

Processes received UDS messages. This function is called by the ssI14229 module with a complete message and is the intended location for the application to handle received messages. This method should not be modified. All subfunctions verify the parameters and call the user-application layer with the message to be processed. Modification should only occur if the default parameter verification is too strict for the implementing user. This may occur if manufacturer specific parameters are being used.

For multi-frame messages this function isn't called until all frames have been received and assembled into a valid and complete message.

This function transmits the positive response assembled in the response buffer if the negative response code has been set to NONE. Otherwise it transmits the negative response.

Parameters:

msg: Pointer to the received message

Return Value:

void

3.6 udsapp_rddataid

Function Prototype:

```
uint8_t
udsapp_rddataid (
        uint16_t dataid,
        uint8_t *data_len,
        uint8_t **buf
)
```

Description:

Read data from specified data id. Data should be packed into buf.

Parameters:

data_id: Data identifier of requested data
data_len: Length of data
buf: Buffer for data to be returned

Return Value:

Output:

Set the data in buf.
Set data_len to length of output.
If data is unavailable at data_id set data_len to zero.

Return:

UDS_NRC_NONE if completed successfully.
UDS_NRC_SAD if data is secured.
UDS_NRC_ROOR if data identifier is invalid.

3.7 udsapp_rdmem

Function Prototype:

```
uint8_t
udsapp_rdmem (
            uint32_t req_addr,
            uint32_t size,
            uint8_t *buf
);
```

Description:

Read data from specified location in memory.

Parameters:

req_addr: Memory location to read
size: Amount of memory to read. Unit is specified in config file
buf: Buffer for data to be returned in

Return Value:

Output:

Set the data in buf.
Set data_len to length of output.
If data is unavailable at req_addr set data_len to zero.

Return:

UDS_NRC_NONE if completed successfully.
UDS_NRC_SAD if data is secured.
UDS_NRC_ROOR if address is invalid.

3.8 udsapp_defdid_id

Function Prototype:

```
uint8_t
udsapp_defdid_id (
        uint16_t data_id,
        uint16_t src_id,
        uint8_t pos,
        uint8_t len
);
```

Description:

Dynamically define new data_id from predefined data_id.

Parameters:

data_id: New data id to be defined
src_id: Existing data id to be referenced
pos: Starting position in src_id's data to be included in data_id
len: Length of data to be included in data id

Return Value:

Return:

UDS_NRC_NONE if completed successfully.
UDS_NRC_SAD if data is secured.
UDS_NRC_ROOR if data identifier is invalid.

3.9 udsapp_defdid_addr

Function Prototype:

```
uint8_t
udsapp_defdid_addr (
        uint16_t data_id,
        uint32_t addr,
        uint32_t size
)
```

Description:

Define data identifier from a memory location.

Parameters:

data_id: New data id to be defined
addr: Memory location to pointed to by data_id
size: Amount of memory to read by data_id.

Return Value:

Return:

UDS_NRC_NONE if completed successfully.
UDS_NRC_ROOR if address is invalid.
UDS_NRC_SAD if address is secured.

3.10 udsapp_cleardid

Function Prototype:

```
uint8_t
udsapp_cleardid (
        uint16_t data_id
)
```

Description:

Clear a dynamically defined data identifer

Parameters:

data_id: Data identifier to be cleared

Return Value:

Return:

UDS_NRC_NONE if completed successfully.
UDS_NRC_SAD if data identifier is secured.
UDS_NRC_ROOR if data identifier is invalid.

3.11 udsapp_write_did

Function Prototype:

```
uint8_t
udsapp_write_did (
          uint16_t data_id,
          uint8_t *data,
          uint16_t len
)
```

Description:

Write the contents of data to location pointed to by data_id.

Parameters:

data_id: Data identifier to be written to
data: Buffer containing data to be written
len: Length of data

Return Value:

Return:

UDS_NRC_NONE if completed successfully.
UDS_NRC_SAD if data identifier is secured.
UDS_NRC_ROOR if data identifier is invalid.

3.12 udsapp_write_mem

Function Prototype:

```
uint8_t
udsapp_write_mem (
        uint32_t addr,
        uint32_t size,
        uint8_t *data
)
```

Description:

Write the contents of data to specified memory address.

Parameters:

addr: Address of data to be read
size: Requested size of data to be read.
data: Buffer containing data to be written

Return Value:

Return:

UDS_NRC_NONE if completed successfully.
UDS_NRC_ROOR if address is invalid.
UDS_NRC_SAD if address is secured.

3.13 udsapp_cleardtc

Function Prototype:

```
uint8_t
udsapp_cleardtc (
          uint32_t dtc
)
```

Description:

Clear state of specified diagnostic trouble code.

Parameters:

dtc: Diagnostic trouble code to be cleared

Return Value:

Return:

UDS_NRC_NONE if completed successfully.
UDS_NRC_ROOR if dtc value is invalid..

3.14 udsapp_iocontrol

Function Prototype:

```
uint8_t
udsapp_iocontrol (
            uint16_t data_id,
            uint8_t *buf,
            uint16_t buf_len,
            uint8_t *rsp,
            uint16_t *rsp_len
)
```

Description:

Control the input or output of data pointed to by data_id. Overrides normal IO of server. buf contains message received by ECU.

Parameters:

data_id: Data identifier to be modified
buf: Buffer containing input message.
buf_len: Length of buf
rsp: Buffer for response
rsp_len: Length of response

Return Value:

Return:

An applicable negative response code or
UDS_NRC_NONE if completed successfully.

3.15 udsapp_routine_start

Function Prototype:

```
uint8_t
udsapp_routine_start (
            uint16_t rout_id,
            uint8_t *control_record,
            uint16_t rec_len,
            uint8_t *rsp,
            uint16_t *rsp_len
)
```

Description:

Starts a predefined routine on the ECU.

Parameters:

rout_id: ID of routine to be started
control_record: Optional parameters for routine
rec_len: Length of parameter record
rsp: Response to be returned to tester
rsp_len: Length of response

Return Value:

Output:

Set rsp and rsp_len.
Set rsp_len to zero if no response to be returned.

Return:

UDS_NRC_NONE if routine started successfully.

3.16 udsapp_routine_stop

Function Prototype:

```
uint8_t
udsapp_routine_stop (
        uint16_t rout_id,
        uint8_t *control_record,
        uint16_t rec_len,
        uint8_t *rsp,
        uint16_t *rsp_len
)
```

Description:

Stop a predefined routine on the ECU.

Parameters:

rout_id: ID of routine to be stopped
control_record: Optional stop parameters for routine
rec_len: Length of parameter record
rsp: Response to be returned to tester
rsp_len: Length of response

Return Value:

Output:

Set rsp and rsp_len.
Set rsp_len to zero if no response to be returned.

Return:

UDS_NRC_NONE if routine stopped successfully.

3.17 udsapp_routine_status

Function Prototype:

```
uint8_t
udsapp_routine_results (
            uint16_t rout_id,
            uint8_t *rsp,
            uint16_t *rsp_len
)
```

Description:

Return status of a completed routine to the tester.

Parameters:

rout_id: ID of routine being queried
rsp: Response to be returned to tester
rsp_len: Length of response

Return Value:

Output:

Set rsp and rsp_len.
Set rsp_len to zero if no response to be returned.

Return:

UDS_NRC_NONE if routine stopped successfully.

3.18 udsapp_reqdl

Function Prototype:

```
uint8_t
udsapp_reqdl (
        uint32_t addr,
        uint32_t size,
        uint8_t fmt
)
```

Description:

Start a download session if one is not in progress.

Parameters:

addr: Starting address for transfer
size: Length of data to be transferred
fmt: Format of data (compression and encryption)

Return Value:

Return:

UDS_NRC_NONE if system is ready for a download.

3.19 udsapp_requl

Function Prototype:

```
uint8_t
udsapp_requl(
        uint32_t addr,
        uint32_t size,
        uint8_t fmt
)
```

Description:

Start an upload session if one is not in progress.

Parameters:

addr: Starting address for transfer
size: Length of data to be transferred
fmt: Format of data (compression and encryption)

Return Value:

Return:

UDS_NRC_NONE if system is ready for an upload.

3.20 udsapp_transfer_data

Function Prototype:

```
uint8_t
udsapp_transfer_data (
        uint8_t bsc,
        uint8_t *buf,
        uint16_t *buf_len
)
```

Description:

Continues an active transfer.

Parameters:

bsc: Block sequence counter
buf: Buffer containing either input data or for output data to be packed into
buf_len: Length of buf

Return Value:

Return:
UDS_NRC_NONE if the packet was handled correctly.

3.21 udsapp_transfer_exit

Function Prototype:

```
uint8_t
udsapp_transfer_exit (
        uint8_t *trpr,
        uint16_t trpr_len,
        uint8_t *rsp,
        uint16_t *rsp_len
)
```

Description:

Requests the exit of a transfer.

Parameters:

trpr: Optional record containing transfer parameters
trpr_len: Length of trpr
rsp: Response to be returned to client
rsp_len: Length of rsp

Return Value:

Return:

UDS_NRC_NONE if transfer was exited.

3.22 udsapp_contr_dtc

Function Prototype:

```
uint8_t
udsapp_contr_dtc (
            uint8_t subfunc,
            uint8_t* opt_rec,
            uint16_t rec_len
)
```

Description:

Control setting of DTCs. Contains an optional user defined record.

Parameters:

subfunc: Subfunction of function
opt_rec: Optional control record
rec_len: Length of record

Return Value:

Return:

UDS_NRC_NONE if successful.
UDS_NRC_ROOR if DTC not supported.

3.24 udsapp_reset_timing

Function Prototype:

```
uint8_t
udsapp_reset_timing (
        void
)
```

Description:

Reset custom timing parameters currently set.

Parameters:

None

Return Value:

Return:

0 on success.
1 on error.

3.25 udsapp_clear_roe

Function Prototype:

```
uint8_t
udsapp_clear_roe (
          void
)
```

Description:

Clear all response on event workflows which have been defined.

Parameters:

None

Return Value:

Return:

0 on success.
1 on error.

3.26 udsapp_reset_security

Function Prototype:

```
uint8_t
udsapp_reset_security (
          void
)
```

Description:

Reset security state to default.

Parameters:

None

Return Value:

Return:

0 on success.
1 on error.

3.27 udsapp_secure_access

Function Prototype:

```
uint8_t
udsapp_secure_access (
            uint8_t sub_func,
            uint8_t *rec,
            uint16_t rec_len,
            uint8_t *rsp,
            uint16_t *rsp_len
)
```

Description:

Exchange seed and key information to change security state of application.

Parameters:

sub_func: Security subfunction being performed
rec: Key record to be read from.
rec_len: Length of rec.
rsp: Buffer for seed to be returned to tester
rsp_len: Length of response.

Return Value:

Output:

Set rsp and rsp_len.
Set rsp_len to zero if no response to be returned.

Return:

UDS_NRC_NONE if message is processed successfully.

3.28 udsapp_session_default

Function Prototype:

```
uint8_t
udsapp_session_default (
          void
)
```

Description:

Transition from the current session to a default session.

Parameters:

void

Return Value:

Return:

UDS_NRC_NONE if processed successfully.

3.29 udsapp_session_programming

Function Prototype:

```
uint8_t
udsapp_session_programming (
          void
)
```

Description:

Transition from current session to programming session.

Parameters:

void

Return Value:

Return:

UDS_NRC_NONE if processed successfully.

3.30 udsapp_session_extended

Function Prototype:

```
uint8_t
udsapp_enable_extended (
          void
)
```

Description:

Transition from current session to extended session.

Parameters:

void

Return Value:

Return:

UDS_NRC_NONE if processed successfully.

3.31 udsapp_session_safety

Function Prototype:

```
uint8_t
udsapp_session_safety (
          void
)
```

Description:

Transition from current session to safety session.

Parameters:

void

Return Value:

Return:

UDS_NRC_NONE if processed successfully.

3.32 udsapp_session

Function Prototype:

```
uint8_t
udsapp_session (
        uint8_t session_type
)
```

Description:

Transition from current session to safety session.

Parameters:

session_type: target session type

Return Value:

Return:

UDS_NRC_NONE if processed successfully.

3.33 udsapp_readdtc

Function Prototype:

```
uint8_t
udsapp_readdtc (
            uint8_t sub_func,
            uint8_t *buf,
            uint16_t buf_len,
            uint8_t *rsp,
            uint16_t *rsp_len
)
```

Description:

Read a diagnostic trouble code

Parameters:

sub_func: Sub-function being requested
buf: Input from tester
buf_len: Length of input buffer
rsp: Buffer for data to be returned
rsp_len: Length of rsp buffer

Return Value:

Output:
    Set rsp and rsp_len.
    Set rsp_len to zero if no response to be returned.
Return:
    UDS_NRC_NONE if message is processed successfully.
    UDS_NRC_SFNS if requested sub-function is not supported.
    UDS_NRC_ROOR if requested DTC is not valid.

3.34 udsapp_rdper_dataid

Function Prototype:

```
uint8_t
udsapp_rdperdataid (
        uint8_t trans_mode,
        uint8_t *dids,
        uint16_t did_cnt,
        uint8_t *rsp,
        uint16_t *rsp_len
)
```

Description:

Read a periodic data identifier.

Parameters:

trans_mode: Transmission mode
dids: Data identifiers to be transmitted
did_cnt: Number of data identifiers
rsp: Buffer for data to be returned
rsp_len: Length of rsp buffer

Return Value:

Output:

Set rsp and rsp_len.
Set rsp_len to zero if no response to be returned.

Return:

UDS_NRC_NONE if message is processed successfully.
UDS_NRC_ROOR if requested data-id is not valid.

Chapter 4: Configuration

- UDS_MAX_RSP_LEN (default 255): Maximum size of response from server application
- UDS_MAX_DID_CNT (default 10): Maximum number of DIDs supported by server
- UDSCFG_TESTER_TIMEOUT_THRESHOLD: Number of ticks which may pass between tester present messages.
- UDSCFG_MAXBLKSIZE: Maximum number of bytes per transfer data request.
- UDSCFG_TRANSFER_FMT: Size of UDSCFG_MAXBLKSIZE in bytes.

These are the configuration parameters which currently are supported. Other parameters will be added as the stack reaches feature completeness.