CiA 317 Final Work Draft

Test interface (COTI) specification

for PC/CAN interfaces

This FDW is for CiA members only and is base for discussion.

Version:0.0.2

11 July 2011

HISTORY

Date	Changes
??.??.2011	*Publication of Version 1.0* as draft standard

NOTE This specification is technically the very same as described formerly in the CiA 301 appendix (clause 6) dated on 2000-01-28.

General information on licensing and patents

CAN in AUTOMATION (CiA) calls attention to the possibility that some of the elements of this CiA specification may be subject of patent rights. CiA shall not be responsible for identifying any or all such patent rights.

Because this specification is licensed free of charge, there is no warranty for this specification, to the extent permitted by applicable law. Except when otherwise stated in writing the copyright holder and/or other parties provide this specification “as is” without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the correctness and completeness of the specification is with you. Should this specification prove failures, you assume the cost of all necessary servicing, repair or correction.

Trademarks

CANopen® and CiA® are registered community trademarks of CAN in Automation. The use is restricted for CiA members or owners of CANopen vendor ID. More detailed terms for the use are available from CiA.

CAN in Automation e. V.
Kontumazgarten 3
DE - 90429 Nuremberg, Germany
Tel.: +49-911-928819-0
Fax: +49-911-928819-79
Url: www.can-cia.org
Email: headquarters@can-cia.org

CONTENTS

1 Scope 4
2 Normative references 4
3 Terms and definitions 4
4 Symbols and abbreviated terms 4
5 COTI functions 4
5.1 General 4
5.2 Function: COTI_InitBoard 5
5.3 Function: COTI_CancelBoard 6
5.4 Function: COTI_ReadBoardInfo“ 7
5.5 Function: COTI_InitCan 8
5.6 Function: COTI_ReadBoardStatus 8
5.7 Function: COTI_ResetBoard 9
5.8 Function: COTI_ReadCanInfo“ 9
5.9 Function: COTI_ReadCanStatus 10
5.10 Function: COTI_ReadObj 11
5.11 Function: COTI_TransmitObj 12
5.12 Function: COTI_RequestObj 12
5.13 Function: COTI_SetTimeout 13
5.14 COTI return-codes 13

1 Scope

This document specifies the programming interface between the CANopen Conformance Test Tool and the low-level driver software of PC/CAN interface modules. The programming interface is independent of the PC interface (DPRAM, LPT, PC-Card, COM) and the used CAN controller chips.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

/CiA301/ CiA 301, CANopen application layer and communication profile

/CiA310-1/ CiA 310-1, CANopen conformance test plan – Part 1: CiA 301 testing

3 Terms and definitions

For the purpose of this document, the following terms and definitions and those given in /CiA301/ and /CiA310-1/ apply.

3.1 active PC/CAN interface module
hardware with local computing power and memory interfacing the PC to the CAN bus lines

3.2 passive PC/CAN interface module
hardware without local computing power interfacing the PC to the CAN bus lines

4 Symbols and abbreviated terms

For the purpose of this document, the following symbols and abbreviated terms and those given in /CiA301/ and /CiA310-1/ apply.

ACR acceptance code register
AMR acceptance mask register
BTR bit-timing register
COTI CANopen test interface
CPU central processing unit
DLL dynamic-link library
FIFO first in, first out
PC personal computer

5 COTI functions

5.1 General

Active and passive PC/CAN interface modules are supported by COTI. When passive PC/CAN interface modules are used the PC manages the CAN controller.

Active PC/CAN interface modules support the PC in the pre-processing of CAN messages as well as in the data storage. This has a positive effect on the processor load of the PC. On the other hand passive PC/CAN interface modules allow the connection of a PC to a CAN network at low costs, but this requires real-time performance of the PC.

The temporary storage of the received messages is done in so-called receive queues. In case of a queue, the messages are stored on their receipt (FIFO concept) and the messages can have different CAN-IDs.

The messages to be sent are written into transmit queues. The micro-controller of the active PC/CAN interface module or an interrupt function of the PC processes produce these messages.

A DLL following the described interface shall have the following features:

- Interface functions defined according to C-calling conventions
- *struct* member alignment is byte
- Only 32-bit DLLs are allowed

5.2 Function: COTI_InitBoard

Function:

```
UINT32 COTI_InitBoard(UINT16 board_seg, COTI_t_UsrIntHdlr fp_int_hdlr,
COTI_t_UsrExHdlr fp_exc_hdlr);
```

Description:

This function registers the given interface in the COTI. This includes resetting the interface and starting the firmware on active PC/CAN interfaces. The access to the hardware is manufacturer-specific and needs to be handled by the implementer. A handle is returned to the PC/CAN interface by which the interface can be addressed. Handles are given as ascending numbers from zero on (0, 1, 2, *to* n).

The function COTI_PrepareBoard shall be executed before the interface is accessed. PC/CAN interfaces already registered and therefore assigned to a program cannot be registered again. (If the PC/CAN interface is used by another application the interface shall be released via COTI_CancelBoard before.)

The call-back handlers are also set with COTI_InitBoard:

- Exception handler for error handling (application specific)
- Receive interrupt handler for the interrupt handling (application specific)

See type definitions of the call-back handlers.

Parameter:

- board_seg (in):
  Adress segment / LPT number / COM number of PC/CAN Interface
- fp_int_hdlr:
  Function pointer to the interrupt function for processing of receive objects. (NULL -> no interrupt processing)

```
/* receive interrupt handler */
typedef void (CALLBACKATTR *COTI_t_UsrRxIntHdlr)( UINT16 count,
                                        COTI_CAN_OBJ FAR * p_obj);
```

  - count:
    Number of objects, stored in the receive queue. (0 -> no object stored)
  - p_obj:
    Pointer to the actual received object to be read


```
typedef struct{
INT32  time_stamp; /*  Time stamp for receive queue objects    */
INT32  id;         /*  Identifier 11-/29-Bit                   */
INT8   len:4;      /*  Number of received data bytes (0-8)     */
INT8   rtr:1;      /*  RTR-Bit: 0= data frame, 1= Remote frame */
INT8   res:3;      /*  reserved                                */
INT8   a_data[8];  /*  Array for the receive data              */
INT8   sts;        /*  Bit coded information for the queue     */
                   /*  Bit0 :  */
                   /*  Bit1 :  */
                   /*  Bit2 :  */
                   /*  Bit3 :  */
                   /*  Bit4 :  */
                   /*  Bit5 :  */
                   /*  Bit6 :  */
                   /*  Bit7 : 1 = receive queue overrun        */
} COTI_CAN_OBJ;
```

- fp_exc_hdlr:
  Function pointer to exception handler for processing of occurred errors. (NULL -> no exception handler)

```
/* exception handler */
typedef void (CALLBACKATTR * COTI_t_UsrExcHdlr)( COTI_FUNC_NUM
                                                 func_num,
                                                 INT32 err_code,
                                                 UINT16 ext_err,
                                                 char * message );
```

  - error_code:
    Internal, vendor specific error code
  - ext_err:
    COTI error code, if possible
    NOTE The 16 LSBs of the 32-bit COTI error code are given.
  - message:
    Terminated error message string

```
typedef enum {
   COTI_PREPARE_BOARD,
   COTI_CANCEL_BOARD,
   COTI_READ_BOARD_INFO,
   COTI_READ_BOARD_STATUS,
   COTI_RESET_BOARD,
   COTI_READ_CAN_INFO,
   COTI_READ_CAN_STATUS,
   COTI_INIT_CAN,
   COTI_READ_OBJ,
   COTI_TRANSMIT_OBJ,
   COTI_REQUEST_OBJ,
   COTI_SET_TIMEOUT
COTI_FUNC_NUM;
```

Return value:

>= 0 -> Board handle
< 0 -> COTI return-codes

5.3 Function: COTI_CancelBoard

Function:

```
INT32 COTI_CancelBoard(UINT32 board_hdl);
```


Description:

This function releases the registered board in the COTI. This includes the reset of the interface and the CAN controllers as well as the de-installation of the used interrupts. The board handle is also released.

Parameter:

- board_hdl (in):
  Handle of a board registered before

Return value:

COTI return-codes

5.4 Function: COTI_ReadBoardInfo“

Function:

```
INT32 COTI_ReadBoardInfo(UINT32 board_hdl , COTI_BOARD_INFO * p_info);
```

Description:

This function reads the board information according to COTI_BOARD_INFO:

~.hw_version	Hardware version as HEX value (i.e.: 0x0100 for V1.00)
~.fw_version	Firmware version as HEX value
~.dd_version	Device driver version as HEX value (only for PC card)
~.sw_version	Version number of PC software as HEX value
~.irq_num	Interrupt number for the communication with the PC/CAN interface
~.board_seg	Configured board address/segment/port number
~.serial_num	16 characters string with the serial number of the board
~.str_hw_type	terminated string with hardware identification

```
typedef struct{
  UINT16 hw_version;      /*  hardware version                    */
  UINT16 fw_version;      /*  firmware Version                    */
  UINT16 dd_version;      /*  device driver version               */
  UINT16 sw_version;      /*  PC software version                 */
  UINT8  irq_num;         /*  used interrupt number               */
  UINT16 board_seg;       /*  used board address                  */
  char   serial_num[16];  /*  String e.g. "Vendor 1234567890"     */
  char   str_hw_type[40]; /*  String e.g. "Testboard V1.00"       */
} COTI_BOARD_INFO;
```

Parameter:

- board_hdl (in):
  Handle of the board registered before
- p_info (out):
  Pointer on the info data

Return value:

COTI return-codes

5.5 Function: COTI_InitCan

Function:

```
INT32 COTI_InitCan(UINT32 board_hdl,  UINT8 baud_rate,  UINT8 mode);
```

Description:

This function initializes the bus timing registers. The values are according to the information given in /CiA301/. The referenced CAN controller is reset, configured with the given baud rate and restarted automatically.

Parameter:

- board_hdl (in):
  Handle of the board registered before
- baud_rate (in):
  Value of the used bit-rate
  0 - 1000 kBit/s
  1 - 800 kBit/s
  2 - 500 kBit/s
  3 - 250 kBit/s
  4 - 125 kBit/s
  5 - 50 kBit/s
  6 - 20 kBit/s
  7 - 10 kBit/s
- mode (in):
  11-bit / 29-bit mode (0 / 1)

NOTE "11-bit" means extended CAN frames are not processed; "29-bit" means normal and extended CAN frames are processed.

Return value:

COTI return-codes

5.6 Function: COTI_ReadBoardStatus

Function:

```
INT32 COTI_ReadBoardStatus(UINT32 board_hdl, COTI_BRD_STS * p_sts);
```

Description:

This function reads the board information according to COTI_BRD_STS:

~.sts Bit coded information of the board status:

Bit 0: RxQueue overrun; an overrun occurred in a configured receive queue (queue was already full and another message could not be entered). Further information can be obtained with COTI_ReadQueStatus and COTI_ReadQueObj.
Bit 4: CAN0-Running
Bit 5: reserved
Bit 6: reserved
Bit 7: reserved

Status bits of the CAN controllers on the board: initialised, started and correctly working CAN controllers are set to ´1´. If the CAN controller is in bus-off status or init mode or if a CAN data overrun or remote queue overrun occurred then the bit is set to ´0´. The exact reason shall then be determined

with COTI_ReadCanStatus. This function allows getting an overview about the actual states of the CAN controllers.

~.cpu_load_average: CPU load in % (0 *to*100)

```
typedef struct{
  UINT8 sts;       /*  Bit coded info of the board state (1 = True)  */
                   /*  Bit0 : RxQueue-Overrun                        */
                   /*  Bit1 :                                        */
                   /*  Bit2 :                                        */
                   /*  Bit3 :                                        */
                   /*  Bit4 : CAN0-Running                           */
                   /*  Bit5 :                                        */
                   /*  Bit6 :                                        */
                   /*  Bit7 :                                        */
  UINT8 cpu_load; /*  Average CPU load in % (0-100)                 */
  } COTI_BRD_STS;
```

Parameter:

- board_hdl (in):
  Handle of the board registered before
- p_sts (out):
  Pointer to the status to be read

Return value:

COTI return-codes

5.7 Function: COTI_ResetBoard

Function:

```
INT32 COTI_ResetBoard(UINT32 board_hdl );
```

Description:

This function resets the PC/CAN interface (software and hardware). The module keeps registered, but communication is interrupted by this. After execution of this function the module and the CAN controllers shall be reinitialized again.

Parameter:

- board_hdl (in):
  Handle of the board registered before

Return value:

COTI return-codes

5.8 Function: COTI_ReadCanInfo“

Function:

```
INT32 COTI_ReadCanInfo(UINT32 board_hdl, COTI_CAN_INFO * p_info);
```

Description:

This function reads the CAN controller type as well as of the configured parameters according to COTI_CAN_INFO:

~.can_type	Type of the CAN controller according to COTI_CAN_TYPE
~.bt0	Configured value for the Bit Timing Register 0
~.bt1	Configured value for the Bit Timing Register 1
~.acc_code	Configured value for the Acceptance Code Register
~.acc_mask	Configured value for the Acceptance Mask Register

```
typedef struct{
  COTI_CAN_TYPE can_type; /* Type of the CAN controller             */
    UINT8  bt0;           /* configured value for the BTR 0         */
    UINT8  bt1;           /* configured value for the BTR 1         */
    UINT32 acc_code;      /* configured value for the ACR           */
    UINT32 acc_mask;      /* configured value for the AMR           */
} COTI_CAN_INFO;

typedef enum{
  COTI_82C200,
  COTI_82527,
  COTI_81C90
} COTI_CAN_TYPE;
```

Parameter:

- board_hdl (in):
  Handle of the board registered before
- p_info (out):
  Pointer to info data

Return value:

COTI return-codes

5.9 Function: COTI_ReadCanStatus

Function:

```
INT32 COTI_ReadCanStatus(UINT32 board_hdl,  COTI_CAN_STS * p_sts);
```

Description:

This function reads the status information of the referenced CAN controller and of the assigned software according to COTI_CAN_STS:

~.sts: Bit coded information of the CAN status (1 = true):

Bit 0: not used

Bit 1: not used

Bit 2: RemoteQueueOverrun - An overrun occurred in the internal queue used for processing of remote requests

Bit 3: CAN-TX-Pending - A transmission operation is just running. If this status lasts without transmitting new data then the CAN controller is not able to send the data (cable break or something similar)

Bit 4: CAN-Init-Mode - CAN is in the initialisation mode

Bit 5: CAN-Data-Overrun - An overrun of CAN messages occurred in the CAN controller (or in the software of the CAN controller)

Bit 6: CAN-Error-Warning-Level - The CAN controller entered the error warning level because of defects on the bus

Bit 7: Bus-off status (the CAN controller transmits recessive state on the bus-lines)

Bits 4 - 7 are directly read from the status registers of the CAN controllers (further information referring these bits can be taken from the data sheets of the CAN controller).

If there is an error in the CAN controller (Bit 2, 5, and 7), then this status shall only be left by executing the function COTI_InitCan.

```
typedef struct{
  UINT8 sts;                  /*  Bit coded information (1 = True)    */
                              /*  Bit0 :                              */
                              /*  Bit1 :                              */
                              /*  Bit2 : RemoteQueue-Overrun          */
                              /*  Bit3 : CAN-TX-Pending               */
                              /*  Bit4 : CAN-Init-Mode                */
                              /*  Bit5 : CAN-Data-Overrun             */
                              /*  Bit6 : CAN-Error-Warning-Level      */
                              /*  Bit7 : CAN-Bus-Off-Status           */
} COTI_CAN_STS;
```

Parameter:

- board_hdl (in):
  Handle of the board registered before
- p_sts (out):
  Pointer to status data

Return value:

COTI return-codes

5.10 Function: COTI_ReadObj

Function:

```
INT32 COTI_ReadObj(UINT32 board_hdl, UINT32 id, UINT16 count,
                                          COTI_CAN_OBJ * p_obj);
```

Description:

This function reads the first entry(ies) of the receive queue. The number of entries to be read are given in ´count´. Only as many entries as available in the queue or supported by the interface are read. This means that the queue shall be read until the value COTI_QUE_EMPTY is returned.

Parameter:

- board_hdl (in):
  Handle of the board registered before
- id (in):
  Identifier of the reading object (filter mode). The id 0xFFFFFFFF performed the reading of the first entry(ies) of the receive queue (queue mode)
- count (in):
  Maximum number of objects which shall be read

- p_obj (out):
  Pointer to the object(s) to be read

Return value:

> 0	Number of read queue entries
= 0	Queue empty (COTI_QUE_EMPTY)
< 0	COTI return-codes

5.11 Function: COTI_TransmitObj

Function:

```
INT32 COTI_TransmitObj(UINT32 board_hdl, UINT32 id, UINT8 len, UINT8 *
                                                              p_data);
```

Description:

This function transmits a CAN data frame via the transmit queue. If COTI_QUE_FULL is returned the referenced transmit queue is actually full and the transmit request shall be repeated (later). If COTI_TX_ERR is returned then the CAN controller cannot transmit messages because of a cable break or incorrect bit-rate.

Parameter:

- board_hdl (in):
  Handle of the board registered before
- id (in):
  Identifier of the transmit object
- len (in):
  Number of data bytes
- p_data (in):
  Pointer to the data to be transmitted

Return value:

COTI return-codes

5.12 Function: COTI_RequestObj

Function:

```
INT32 COTI_RequestObj(UINT32 board_hdl, UINT32 id, UINT8 len, UINT8 *
                                                              p_data);
```

Description:

This function transmits a CAN remote frame via the transmit queue and wait for the response. If COTI_QUE_FULL is returned the referenced transmit queue is actually full and the transmitting order shall be repeated (later). If COTI_TX_ERR is returned the CAN controller cannot transmit messages because of a cable break or incorrect baudrate.

Parameter:

- board_hdl (in)
  Handle of the board registered before

- id (in)
  Identifier of the transmit object
- len (in):
  Number of data bytes
- p_data (out):
  Pointer to the data to be request

Return value:

COTI return-codes

5.13 Function: COTI_SetTimeout

Function:

```
INT32 COTI_SetTimeout(UINT32 board_hdl, UINT32 time_out);
```

Description:

This function sets the maximum time-out time for the COTI functions.

Parameter:

- board_hdl (in):
  Handle of the board registered before
- time_out (in):
  Maximum time out time in ms

Return value:

COTI return-codes

5.14 COTI return-codes

No other than the specified COTI return-codes shall be used.

```
/*
**  COTI-Return-Codes:
**  A defined exception handler will additionally be called !!!
*/

#define  COTI_ERR         -255    /*  unknown error                    */
#define  COTI_TIME_OUT    -254    /*  Time out                         */
#define  COTI_QUE_EMPTY   -253    /*  Rx-Queue empty                   */
#define  COTI_QUE_FULL     0      /*  Tx-Queue full                    */
#define  COTI_OK           1      /*  No error occurred                */
#define  COTI_HWSW_ERR    -1      /*  hardware or software error       */
#define  COTI_SUPP_ERR    -2      /*  Function not supported           */
#define  COTI_PARA_ERR    -3      /*  Parameter error, wrong value     */
#define  COTI_RES_ERR     -4      /*  Resource error, no memory left   */
#define  COTI_QUE_ERR     -5      /*  Queue overrun, data lost         */
#define  COTI_TX_ERR      -6      /*  transmission error               */
```

