

Product / Project: iID[®] (mobile) driver engine

Customer / Project Code: -

Product: MICROSENSYS iID[®]3000
RFID interfaces

Product Code: -

Revision: 0x10.0xC

Date: 2007-09-06

API - Definition for Windows[®] mobile and PC Windows[®]

This documents describes the software API of both iID[®] 3000 PRO driver engine for MICROSOFT Windows[®] based PC systems and iID[®] mobile driver engine for MICROSOFT Pocket-PC 3.0, Pocket-PC 2002, Pocket-PC 2003, Windows[®] Mobile 5.0, Windows[®] CE.Net 4.x,5.0, based on ARMV4 or X86 CPU.

Drivename: iiddrv30_pro.dll
Tested devices: MICROSOFT[®] Pocket-PC 2003, Windows[®] Mobile 5.0,
Windows[®] CE.Net 4.2,5.0
Supported host interface: RS232 serial, CompactFlash, Bluetooth SPP
Version: 0x10.0xC
Release date: 2007-09-06

Drivename: iiddrv30_pro.dll
Tested devices: MICROSOFT[®] Windows XP/SP2
Supported host interface: RS232 serial, USB, Bluetooth SPP
Version: 0x10.0xB
Release date: 2007-07-12

Charset: Windows[®] ANSI charset

Contents

Short history	2
Common port functions	3
Common RFID interface functions	5
iID [®] (mobile) driver functions	7
ISO15693 TAG functions	12
iID [®] -G TAG functions	15
iID [®] -L TAG memory functions	16
iID [®] -L TAG sensor functions	20
I-Code [®] UID TAG functions	23
I-Code [®] 1 TAG functions	25
iID [®] -D TAG functions (preliminary)	27
LEGIC [®] Prime functions (preliminary)	28
Mifare UltraLight [™] functions (preliminary)	29
125/134 kHz system TAG functions (preliminary)	30
Error - Codetable	32

1. Short history

This chapter includes a short history of modifications of iiddrv30_pro.dll.

Date	Reason	Modification	Release Date / Version	FileName
2007-05	- first release based on iID [®] 2000 driver engine		0x10.0xA	iiddrv30_pro.dll
2007-07	- reworked iID [®] functions to support high memory transponders	- c_readbytes - c_writebytes		
	- support for additional transponders	- added preliminary support for Mifare UltraLight transponders		
2007-07-16	- support for microsensys sensor functions		0x10.0xB	iiddrv30_pro.dll
2007-08-30	- added support for microsensys TELID [®] 242 sensor transponder	- added: telid242_c_get_status, telid242_c_get_calibration, telid242_c_get_temperature, telid242_c_get_pressure	0x10.0xC	iiddrv30_pro.dll (only PC / WIN32)

2. Common port functions

This chapter includes functions, which are necessary for configuration of driver and host-side peripheral interface.

HANDLE __stdcall APIENTRY __stdcall c_openinterface (LPTSTR lpszPortName);

Parameter : lpszPortName - name of serial port, where RFID interface is connected (e.g. "COM2"), no significance for CompactFlash- or USB-mode
 Result : success - handle of serial port
 error - INVALID_HANDLE_VALUE
 Description : function generates handle to serial, CompactFlash, USB or Bluetooth™ serial port and sets RFID interface parameters

BYTE __stdcall APIENTRY __stdcall c_closeinterface (HANDLE m_HCom);

Parameter : m_HCom - handle of port opened with c_openinterface
 Result : success - 0
 error - 1
 Description : function closes handle to serial, CompactFlash, USB or Bluetooth™ serial port

HANDLE __stdcall APIENTRY __stdcall c_get_handle ();

Parameter : -
 Result : success - handle of port created with c_initialize();
 error – INVALID_HANDLE_VALUE
 Description : function returns handle to serial, CompactFlash, USB or Bluetooth™ serial port created with c_initialize() or INVALID_HANDLE_VALUE

BYTE __stdcall APIENTRY __stdcall c_set_handle (HANDLE m_HCom);

Parameter : HANDLE m_HCom
 Result : success - 0
 Description : function overwrites/sets handle for iID® driver functions

HANDLE __stdcall APIENTRY __stdcall c_get_port_state (HANDLE m_HCom);

Parameter : m_HCom - handle of port opened with c_openinterface
 Result : success - handle of port
 error – INVALID_HANDLE_VALUE
 Description : function checks handle to serial, CompactFlash, USB or Bluetooth™ serial port and returns valid handle or INVALID_HANDLE_VALUE

BYTE __stdcall APIENTRY __stdcall c_set_port_type (BYTE porttype, LPTSTR lpszPortName);

Parameter : lpszPortName - name of serial port, where RFID interface is connected (standard "COM2"), no significance for CompactFlash- or USB-mode
 porttype : - 0 – serial standard interface (CF or USB in serial mode)
 - 1 – Compact-Flash interface (no serial emulation, only mobile platforms)
 - 2 – Bluetooth™ serial standard interface
 - 4 – USB interface (no serial emulation, only PC platforms)
 Result : success - porttype
 error – 0xFF
 Description : function sets port parameters for c_initialize and c_openinterface

BYTE __stdcall APIENTRY __stdcall c_set_interface_type(INT32 frequency);

Parameter	:	frequency	- 0x7D/125 – 125/134kHz RFID interface mode - 0x54C/1356 – 13.56MHz RFID interface mode
Result	:	success - 0 error – 0xFF	
Description	:	function sets port parameters for c_initialize and c_openinterface and other iID [®] driver functions.	

3. Common RFID interface functions

This chapter describes functions available for communication and configuration of the RFID interface without necessary TAG communication.

BYTE __stdcall APIENTRY __stdcall c_reader_sleep_on_iid3000 (HANDLE m_HCom);

Parameter : m_HCom - handle of port, where RFID interface is connected
 Result : success - 0
 error – see error code table
 Description : function sets RFID interface into power-save mode

BYTE __stdcall APIENTRY __stdcall iid3000_c_read_reader_id (HANDLE m_HCom, INT32 *reader_id, BYTE *pByteArray);

BYTE __stdcall APIENTRY __stdcall c_read_reader_id_iid3000 (HANDLE m_HCom, INT32 *reader_id, BYTE *pByteArray);

Parameter : m_HCom – handle of port with RFID interface
 *reader_id – pointer to INT32, which contains interface-ID after successful completion of instruction
 *pByteArray – pointer to array of byte, which contains interface data after successful completion of instruction, byte 0 contains length of data
 Result : success - 0
 error – see error table
 Description : function reads ID-number and parameters from RFID interface with iID@3000 protocol frame

BYTE __stdcall APIENTRY __stdcall c_set_mode_125 (HANDLE m_HCom, BYTE Param);

Parameter : m_HCom – handle of port with RFID interface
 Param - byte with control information for RFID interface for handling different 125kHz transponders (ask MICROSENSYS for further information)
 Result : success - 0
 error – see error table
 Description : function sets port parameters for 125kHz RFID operation mode of interface.

BYTE __stdcall APIENTRY __stdcall iso15693_set_optionflag (HANDLE m_HCom, BYTE flag);

Parameter : m_HCom – handle of port with RFID interface
 flag – byte with control information for RFID interface for handling different ISO15693 transponders (ask MICROSENSYS for further information)
 Result : success - 0
 error – see error table
 Description : function sets parameters for RFID interface

BYTE __stdcall APIENTRY __stdcall c_led_off_iid3000 (HANDLE m_HCom);

Parameter : m_HCom – handle of port with RFID interface
Result : success - 0
 error – see error table
Description : function switches ON LED of RFID interface
 [This command is only available on devices with LED support.](#)

BYTE __stdcall APIENTRY __stdcall c_led_on_iid3000 (HANDLE m_HCom);

Parameter : m_HCom – handle of port with RFID interface
Result : success - 0
 error – see error table
Description : function switches OFF LED of RFID interface
 [This command is only available on devices with LED support.](#)

4. iID® (mobile) driver functions

The following functions describe functions, which are hardware independent and abstracted from different transponder systems based on system-specific functions described in the next chapters. Using these functions is more comfortable, but sometimes also time-expensive, so in time-critical processes the usage of system-specific functions is recommended.

BYTE __stdcall APIENTRY __stdcall c_initialize();

Parameter :
 Result : returns 0 if successful, or error-code (see error table)
 Description : Instruction scans for MICROSENSYS interface, creates handle to the hardware-device and initializes device (appr. 30ms Power-ON-time included). The (mobile) device should NOT be turned off before calling c_terminate().

BYTE __stdcall APIENTRY __stdcall c_terminate ();

Parameter :
 Result : returns 0 if port successfully closed or error-code (see error table)
 Description : Instruction closes interface-connection and device handle. Device is set into power-optimized mode.

BYTE __stdcall APIENTRY __stdcall c_read_reader_id (INT32 *reader_id, BYTE *piIDByteArray);

Parameter : *m_Hcom* - handle of CompactFlash port to be worked with
 *reader_id - pointer to integer for internal id-number of reader
 pByteArray - address of buffer to store data in
 Result : returns 0 if successful, or error-code (see error table)
 PData:

db0	db1	db2	db3	db4	db5	db6	db7
ID(low)	ID(High)	internal	hardware-config	hardware-index	firmware-config	firmware-index	reserved

Description : Command reads Reader_ID from interface connected to host.

BYTE __stdcall APIENTRY __stdcall c_identify (BYTE *piIDByteArray);

Parameter : pByteArray - address of buffer to store data in, returns (byte 0 -> length of data, byte 1..8 -> data)
 Result : returns 0 if successful, or error-code (see error table)
 Description : Command is reading UniqueIdentifier from any TAG near antenna of the interface, it includes an auto scanning feature for all known transponders. Scanning timeout is defined by c_set_timeout, standard is 1000msec.

BYTE __stdcall APIENTRY __stdcall c_identify_all (BYTE *pIDByteArray, BYTE Hold, BYTE *pCompare);

Parameter : pByteArray - address of buffer to store data in, returns (byte 0 -> number of identifiers (n), byte 1..(n*8) -> data)
 Hold = 0 – search for all transponders in communication range
 Hold = 1 – search for transponder with special UID and stop scanning, when found (Hold-Mode)
 *pCompare – pointer to array of byte, which contains identifier to search for (only necessary for Hold-Mode)
 Result : returns 0 if successful, or error-code (see error table)
 Description : Command is reading UniqueIdentifier from multiple TAGs near antenna of the Interface (anticollision mode), it includes an auto scanning feature for all known transponders.
 Scanning timeout is defined by c_set_timeout, standard is 1000msec.

BYTE __stdcall APIENTRY __stdcall c_get_transponder_parameters(INT32 *ptagtype, INT32 *pmaxlength, INT32 *tagsystem, INT32 *pxtraparam, BYTE *pIDByteArray); (preliminary)

Parameter : *ptagtype - pointer to int32 containing transponder type information
 *pmaxlength - pointer to int32 containing maximum read/write accessible transponder memory (in bytes)
 *ptagtype - pointer to int32 containing transponder system information
 Other parameters are for future use!
 Result : returns 0 if successful
 Description : Command informs about tag parameters of previous successful c_identify, c_identify_all.

BYTE __stdcall APIENTRY __stdcall c_readbytes (BYTE identifier[], INT32 from, INT32 length, BYTE *pIDByteArray);

Parameter : from - TAG start address (0..n) to read data from (for interface type 0x10 : from should be a multiple of 8)
 length - length (m) of data to read from TAG (for interface type 0x10 : length should be a multiple of 8)
 pByteArray - address of buffer to store data in (byte 0 -> length of data, byte 1..m -> data)
 pIdentifier - address of buffer with identifier (byte[0..7] -> data, see c_Identifier() for more details)
 Result : returns 0 if successful, or error-code (see error table)
 returns (byte 0 -> length of data, byte 1..m -> data)
 Description : Command is reading m bytes from any transponder near receiver, it includes an auto scanning feature for all known transponders.
 Command uses anticollision mode on prior calling c_identify_all.
 Scanning timeout is defined by c_set_timeout, standard is 1000msec. Please note, that although possible, the maximum packet length should be <=512 bytes.

BYTE __stdcall APIENTRY __stdcall c_writebytes (BYTE identifier[],INT32 from,INT32 length, BYTE *pByteArray, BYTE lock);

Parameter : *from* - TAG start address (0..n) to write data to (for interface type 0x10 : *from* should be a multiple of 8)
length - length (m) of data to read from TAG (for interface type 0x10 : *length* should be a multiple of 8)
pByteArray - address of buffer with data to be stored (byte[0..length-1])
pldentifier - address of buffer with identifier (byte[0..7] -> data, see c_Identify() for more details)
lock - true, if blocks should be locked after writing, otherwise false (**Attention: locking the blocks is irreversible and blockwise! See transponder chip documentation for more information**)

Result : returns 0 if successful, or error-code (see error table)

Description : Command is writing m bytes to any transponder near receiver, it includes an auto scanning feature for all known transponders..
 Command uses anticollision mode on prior calling c_identify_all.
 Scanning timeout is defined by c_set_timeout, standard is 1000msec. Please note, that although possible, the maximum packet length should be <=512 bytes.

BYTE __stdcall APIENTRY __stdcall c_get_temperature (double *temperature, BYTE *pByteArray);

Parameter : *m_Hcom* - handle of CompactFlash port to be worked with
 *temperature - pointer to float for temperature data
 pByteArray - address of buffer to store data in

Result : returns 0 if successful

Description : Command is reading the temperature from any TAG supporting this feature near receiver.

BYTE __stdcall APIENTRY __stdcall c_emul_on();

Parameter :

Result : returns 0 if successful

Description : Command is setting driver into software emulation mode (no hardware response needed).

BYTE __stdcall APIENTRY __stdcall c_emul_off();

Parameter :

Result : returns 0 if successful

Description : Command is setting driver into hardware communication mode (standard).

BYTE __stdcall APIENTRY __stdcall c_tag_appeared();

Parameter :

Result : returns 0 if successful

Description : If driver is in software emulation mode, tag-operations seem to be successful. Driver response is some dummy-data.

BYTE __stdcall APIENTRY __stdcall c_tag_disappeared();

Parameter :
 Result : returns 0 if successful
 Description : If driver is in software emulation mode, tag-operations seem to be erroneous. Driver response is like no tag near antenna.

BYTE __stdcall APIENTRY __stdcall c_set_timeout (INT32 newtimeout);

Parameter : *newtimeout* – new timeout for c_identify, c_readbytes and c_writebytes in msec, value should be between 1 and 4999
 Result : returns 0 if successful
 Description : Command is setting a new scanning timeout for c_identify, c_readbytes and c_writebytes.

INT32 __stdcall APIENTRY __stdcall c_get_timeout ();

Parameter :
 Result : returns actual scanning timeout
 Description : Command is reading the actual scanning timeout for c_identify, c_readbytes and c_writebytes

BYTE __stdcall APIENTRY __stdcall c_set_system_mask (INT32 new_mask);

Parameter : *new_mask* – new mask for transponder systems, which are handled by iID[®] driver functions (multiple selection available), standard: all systems supported

0x1	– ISO 15693 /125kHz UNIQUE transponders enabled/disabled and/or
0x2	– iID [®] -L /125kHz TITAN transponders enabled/disabled and/or
0x4	– iID [®] -D /125kHz PSK system enabled/disabled and/or
0x8	– iID [®] -G - system enabled/disabled and/or
0x10	– ARI0 64bit /134kHz FDX-B system enabled/disabled
0x20	– I-Code UID - system enabled/disabled
0x40	– LEGIC Prime UID - system enabled/disabled and/or
0x80	– I-Code 1 - transponders enabled/disabled and/or
0x100	– Mifare Ultralight - transponders enabled/disabled

Result : returns 0 if successful, or error-code (see error table)
 Description : Command is setting the scanning mask of iID[®] driver functions for different supported transponder systems and may be used for decreasing the operation time.

INT32 __stdcall APIENTRY __stdcall c_get_system_mask();

Parameter :
 Result : returns actual system mask for iID[®] driver functions
 Description : Command is reading iID[®] driver parameter for handled transponder systems.

BYTE __stdcall APIENTRY __stdcall c_get_driver_version (INT32 *main_version, INT32 *sub_version);

Parameter	:	*main_version – pointer to INT32, which contains main version of DLL
	:	*sub_version – pointer to INT32, which contains sub version of DLL
Result	:	success - 0
Description	:	function reads version number of DLL for application programmer support

5. ISO15693 TAG functions

This chapter describes functions available for communication with transponders based on the standard ISO15693.

BYTE __stdcall APIENTRY __stdcall iso15693_read_uid (HANDLE m_HCom, BYTE *pByteArray);

Parameter : m_HCom – handle of port with RFID interface
 *pByteArray – pointer to array of byte, which contains interface data after successfull completion of instruction, byte 0 contains length of data

Result : success - 0
 error – see error table

Description : function reads UID from ISO15693 transponder near RFID interface

BYTE __stdcall APIENTRY __stdcall iso15693_read_uid_all (HANDLE m_HCom, BYTE *pNrOfFound, BYTE *pByteArray, BYTE Hold, BYTE *pCompare);

Parameter : m_HCom – handle of port with RFID interface
 *pNrOfFound – pointer to byte, which contains number of identifiers found in communication range after successful completion
 *pByteArray – pointer to array of byte, which contains interface data after successfull completion of instruction, byte 0 contains length of data
 Hold = 0 – search for all transponders in communication range
 Hold = 1 – search for transponder with special UID and stop scanning, when found (Hold-Mode)
 *pCompare – pointer to array of byte, which contains identifier to search for (only necessary for Hold-Mode)

Result : success - 0
 error – see error table

Description : function reads UID from ISO15693 transponder near RFID interface in Anticollision mode

BYTE __stdcall APIENTRY __stdcall iso15693_read_block (HANDLE m_HCom, INT32 BlockAddress, BYTE *pByteArray);

Parameter : m_HCom – handle of port with RFID interface
 BlockAddress – address of transponder to read data from
 *pByteArray – pointer to array of byte, which contains interface data after successfull completion of instruction, byte 0 contains length of data

Result : success - 0
 error – see error table

Description : function reads block from ISO15693 transponder near RFID interface

BYTE __stdcall APIENTRY __stdcall iso15693_read_block_addressed (HANDLE m_HCom, BYTE *pIdentifier, INT32 BlockAddress, BYTE *pByteArray);

Parameter : m_HCom – handle of port with RFID interface
 BlockAddress – address of transponder to read data from
 *pIdentifier – pointer to array of byte, which contains identifier of transponder to select for read/write operation
 successfull completion of instruction, byte 0 contains length of data
 *pByteArray – pointer to array of byte, which contains interface data after successfull completion of instruction, byte 0 contains length of data

Result : success - 0
 error – see error table

Description : function reads block from ISO15693 transponder near RFID interface in anticollision mode

BYTE __stdcall APIENTRY __stdcall iso15693_write_block (HANDLE m_HCom, INT32 BlockAddress, BYTE *pByteArray);

Parameter : m_HCom – handle of port with RFID interface
 BlockAddress – address of transponder to write data to
 *pByteArray – pointer to array of byte, which contains data to write to transponder

Result : success - 0
 error – see error table

Description : function writes block to ISO15693 transponder near RFID interface

BYTE __stdcall APIENTRY __stdcall iso15693_write_block_addressed (HANDLE m_HCom, BYTE *pIdentifier, INT32 BlockAddress, BYTE *pByteArray);

Parameter : m_HCom – handle of port with RFID interface
 BlockAddress – address of transponder to write data to
 *pIdentifier – pointer to array of byte, which contains identifier of transponder to select for read/write operation
 *pByteArray – pointer to array of byte, which contains data to write to transponder

Result : success - 0
 error – see error table

Description : function writes block to ISO15693 transponder near RFID interface in anticollision mode

BYTE __stdcall APIENTRY __stdcall iso15693_lock_block (HANDLE m_HCom, INT32 BlockAddress);

Parameter : m_HCom – handle of port with RFID interface
 BlockAddress – address of transponder to lock

Result : success - 0
 error – see error table

Description : function locks block of ISO15693 transponder near RFID interface (Att.: This command is irreversible !)

BYTE __stdcall APIENTRY __stdcall iso15693_set_myd_custom_mode(BYTE New_Mode);

Parameter : New_Mode – 0 -> use ISO optional commands READ and WRITE (standard),
1 -> use my-D custom commands READ and WRITE
Result : success - 0
error – see error table
Description : function switches driver functionality for readbytes(), writebytes() and
ISO15693 TAG functions (special function my-D, not necessary for other
transponders)

BYTE __stdcall APIENTRY __stdcall iso15693_get_myd_custom_mode();

Parameter :
Result : 0 -> use ISO optional commands READ and WRITE
1 -> use my-D custom commands READ and WRITE
Description : function returns driver parameter for my-D commandset

***BYTE __stdcall APIENTRY __stdcall iso15693_myd_write_byte (HANDLE m_HCom,INT32
BlockAddress, BYTE Position, BYTE Data);***

Parameter : m_HCom – handle of port with RFID interface
BlockAddress – address of transponder to write data to
Position – byte offset within page of transponder to write data to
Data – data byte to write
Result : success - 0
error – see error table
Description : function writes single byte to my-D transponder near RFID interface
(command is only available for my-D in custom mode)

6. iID®-G TAG functions

This chapter describes functions available for communication with transponders based on the MICROSENSYS RFID system iID®-G.

BYTE __stdcall APIENTRY __stdcall iidg_c_read_uid (HANDLE m_HCom, BYTE *pByteArray);

Parameter : m_HCom – handle of port with RFID interface
 *pByteArray – pointer to array of byte, which contains interface data after successful completion of instruction, byte 0 contains length of data, bytes 7..8 are not part of UID and contain manufacturer information

Result : success - 0
 error – see error table

Description : function reads UID from iID®-G transponder near RFID interface

BYTE __stdcall APIENTRY __stdcall iidg_c_read_ro (HANDLE m_HCom, BYTE *pByteArray);

Parameter : m_HCom – handle of port with RFID interface
 *pByteArray – pointer to array of byte, which contains interface data after successful completion of instruction, byte 0 contains length of data

Result : success - 0
 error – see error table

Description : function reads customer programmable ReadOnly-Code from iID®-G transponder near RFID interface

BYTE __stdcall APIENTRY __stdcall iidg_c_read_block_16 (HANDLE m_HCom, INT32 BlockAddress, BYTE param, BYTE *pByteArray);

Parameter : m_HCom – handle of port with RFID interface
 BlockAddress – address of transponder to read data from
 param – parameter byte, for future development, should be 0
 *pByteArray – pointer to array of byte, which contains interface data after successful completion of instruction, byte 0 contains length of data

Result : success - 0
 error – see error table

Description : function reads block from iID®-G transponder near RFID interface

BYTE __stdcall APIENTRY __stdcall iidg_c_write_block_16 (HANDLE m_HCom, INT32 BlockAddress, BYTE param, BYTE *pByteArray);

Parameter : m_HCom – handle of port with RFID interface
 BlockAddress – address of transponder to write data to
 param – parameter byte, for future development, should be 0
 *pByteArray – pointer to array of byte, which contains data to write to transponder

Result : success - 0
 error – see error table

Description : function writes block to iID®-G transponder near RFID interface

BYTE __stdcall APIENTRY __stdcall iidg_c_set_lockbit (HANDLE m_HCom);

Parameter : m_HCom – handle of port with RFID interface

Result : success - 0
 error – see error table

Description : function sets lockbit for customer OTP area of iID®-G transponder (see transponder description) (**Attention: operation is irreversible!**)

7. iID[®]-L memory functions

int __stdcall APIENTRY __stdcall iidl_c_read_ro_code (HANDLE m_HCom, BYTE *pByteArray);

Parameter : m_HCom – handle of port with RFID interface
 *pByteArray – pointer to array of byte, which contains interface data after
 successfull completion of instruction, byte 0 contains length of data

Result : success - 0
 error – see error table

Description : function reads ReadOnly-Code from iID-L transponder near RFID interface

int __stdcall APIENTRY __stdcall iidl_c_write_ro_code (HANDLE m_HCom, BYTE *pByteArray, BYTE lock)

Parameter : m_HCom – handle of port with RFID interface
 *pByteArray – pointer to array of byte, which contains data to write to
 transponder
 lock <> 0 – set lock function for defined data (irreversible)

Result : success - 0
 error – see error table

Description : function writes customer area of ReadOnly-Code (4Byte) to iID-L transponder
 near RFID interface

int __stdcall APIENTRY __stdcall iidl_c_write_block (HANDLE m_HCom, unsigned int BlockAddress, BYTE *pByteArray, BYTE lock);

Parameter : m_HCom – handle of port with RFID interface
 BlockAddress – address of transponder to write data to
 *pByteArray – pointer to array of byte, which contains data to write to
 transponder
 lock <> 0 – set lock function for defined data (irreversible)

Result : success - 0
 error – see error table

Description : function writes block (4Byte) to iID-L transponder near RFID interface

int __stdcall APIENTRY __stdcall iidl_c_read_block (HANDLE m_HCom, unsigned int BlockAddress, BYTE *pByteArray);

Parameter : m_HCom – handle of port with RFID interface
 BlockAddress – address of transponder to read data from
 *pByteArray – pointer to array of byte, which contains interface data after
 successfull completion of instruction, byte 0 contains length of data

Result : success - 0
 error – see error table

Description : function reads block (4Byte) from iID-L transponder near RFID interface

int __stdcall APIENTRY __stdcall iidl_c_write_protect_code (HANDLE m_HCom, BYTE *pOldPw, BYTE *pPwArray);

Parameter : m_HCom – handle of port with RFID interface
 BlockAddress – address of transponder to write data to
 *pOldPw – pointer to array of byte, which contains actual password
 *pPwArray – pointer to array of byte, which contains new password

Result : success - 0
 error – see error table

Description : function sets new protect-code for iID-L transponder near RFID interface

int __stdcall APIENTRY __stdcall iidl_c_set_protection (HANDLE m_HCom, BYTE param);

Parameter : m_HCom – handle of port with RFID interface
 BlockAddress – address of transponder to write data to
 param – enable/disable protect function
 param=0 – disable protect function
 param=1 – enable protect function

Result : success - 0
 error – see error table

Description : function enables/disables protect function of iID-L transponder near RFID interface

int __stdcall APIENTRY __stdcall iidl_c_send_protect_code (HANDLE m_HCom, BYTE *pPwArray);

Parameter : m_HCom – handle of port with RFID interface
 BlockAddress – address of transponder to write data to
 *pPwArray – pointer to array of byte, which contains password

Result : success - 0
 error – see error table

Description : function enables access for iID-L transponder near RFID interface with activated protect function

int __stdcall APIENTRY __stdcall iidl_c_read_block_ee (HANDLE m_HCom, int BlockAddress, BYTE* pByteArray);

Parameter : m_HCom – handle of port with RFID interface
 BlockAddress – address of transponder to read data from
 *pByteArray – pointer to array of byte, which contains interface data after successful completion of instruction, byte 0 contains length of data

Result : success - 0
 error – see error table

Description : function reads block (16Byte) from E²PROM of iID-L transponder near RFID interface

int __stdcall APIENTRY __stdcall iidl_c_write_block_ee (HANDLE m_HCom,int BlockAddress, BYTE *pByteArray);

Parameter : m_HCom – handle of port with RFID interface
BlockAddress – address of transponder to write data to
*pByteArray – pointer to array of byte, which contains data to write to transponder

Result : success - 0
error – see error table

Description : function writes block (16Byte) to E²PROM of iID-L transponder near RFID interface (usage of command is depending on transponder configuration)

int __stdcall APIENTRY __stdcall iidl_c_read_block_ee_64 (HANDLE m_HCom,int BlockAddress, BYTE* pByteArray)

Parameter : m_HCom – handle of port with RFID interface
BlockAddress – address of transponder to read data from
*pByteArray – pointer to array of byte, which contains interface data after successfull completion of instruction, byte 0 contains length of data

Result : success - 0
error – see error table

Description : function reads block (64Byte) from E²PROM of iID-L transponder near RFID interface

int __stdcall APIENTRY __stdcall iidl_c_write_block_ee_64 (HANDLE m_HCom,int BlockAddress, BYTE *pByteArray)

Parameter : m_HCom – handle of port with RFID interface
BlockAddress – address of transponder to write data to
*pByteArray – pointer to array of byte, which contains data to write to transponder

Result : success - 0
error – see error table

Description : function writes block (64Byte) to E²PROM of iID-L transponder near RFID interface (usage of command is depending on transponder configuration)

int __stdcall APIENTRY __stdcall iidl_c_read_block_ee_128 (HANDLE m_HCom,int BlockAddress, BYTE* pByteArray)

Parameter : m_HCom – handle of port with RFID interface
BlockAddress – address of transponder to read data from
*pByteArray – pointer to array of byte, which contains interface data after successfull completion of instruction, byte 0 contains length of data

Result : success - 0
error – see error table

Description : function reads block (128Byte) from E²PROM of iID-L transponder near RFID interface

int __stdcall APIENTRY __stdcall iidl_c_write_block_ee_128 (HANDLE m_HCom,int BlockAddress,BYTE *pByteArray)

Parameter	:	m_HCom – handle of port with RFID interface BlockAddress – address of transponder to write data to *pByteArray – pointer to array of byte, which contains data to write to transponder
Result	:	success - 0 error – see error table
Description	:	function writes block (128Byte) to E ² PROM of iID-L transponder near RFID interface (usage of command is depending on transponder configuration)

8. iID[®]-L sensor functions

int **__stdcall APIENTRY __stdcall iidl_c_read_in2 (HANDLE m_HCom, BYTE *setting)**

Parameter : m_HCom – handle of port with RFID interface
 *setting – pointer to byte, which contains switch data after command completion
 bit 0=0 – IN1 open
 bit 0=1 – IN1 closed
 bit 1=0 – IN2 open
 bit 1=1 – IN2 closed

Result : success - 0
 error – see error table

Description : function reads switch data of iID-L transponder near RFID interface
 (usage of command is depending on transponder configuration)

int **__stdcall APIENTRY __stdcall iidl_c_set_out2 (HANDLE m_HCom, BYTE setting);**

Parameter : m_HCom – handle of port with RFID interface
 setting –byte, which contains switch data for command completion
 bit 1=0 – OUT1 open
 bit 1=1 – OUT1 closed
 bit 2=0 – OUT2 open
 bit 2=1 – OUT2 closed

Result : success - 0
 error – see error table

Description : function sets switch data of iID-L transponder near RFID interface as long as
 interface antenna is active
 (usage of command is depending on transponder configuration)

int **__stdcall APIENTRY __stdcall telid2io_c_read_in (HANDLE m_HCom, BYTE *setting);**

Parameter : m_HCom – handle of port with RFID interface
 *setting – pointer to byte, which contains switch data after command completion
 bit 6=0 – IN1 open
 bit 6=1 – IN1 closed
 bit 5=0 – IN2 open
 bit 5=1 – IN2 closed
 bit 4=0 – IN3 open
 bit 4=1 – IN3 closed
 bit 3=0 – IN4 open
 bit 3=1 – IN4 closed

Result : success - 0
 error – see error table

Description : function reads switch data of iID-L transponder near RFID interface
 (usage of command is depending on transponder configuration)

int __stdcall APIENTRY __stdcall telid2io_c_set_out (HANDLE m_HCom, BYTE setting);

Parameter : m_HCom – handle of port with RFID interface
 setting –byte, which contains switch data for command completion
 bit 7=0 – OUT1 open
 bit 7=1 – OUT1 closed
 bit 8=0 – OUT2 open
 bit 8=1 – OUT2 closed

Result : success - 0
 error – see error table

Description : function sets switch data of iLD-L transponder near RFID interface as long as interface antenna is active
 (usage of command is depending on transponder configuration)

int __stdcall APIENTRY __stdcall c_get_temperature_telid21x (HANDLE m_HCom, double *temp, BYTE *pByteArray)

Parameter : m_Hcom - handle of port to be worked with
 *temperature - pointer to float for temperature data
 pByteArray - address of buffer to store data in

Result : returns 0 if successfull

Description : Command is reading the temperature from any iLD-L transponder near receiver.

int __stdcall APIENTRY __stdcall telid242_c_get_status (HANDLE m_HCom, int param, BYTE *pByteArray, BYTE *pCode)
preliminary (Ver 0x10.0xC)

Parameter : m_Hcom - handle of port to be worked with
 pByteArray - address of buffer to store data in (3 bytes,byte[0] contains length)
 param - parameter byte for additional function control

Bit	Value	Description
0	0	Turn ON RF-field and DELAY prior operation (required for the first TELID242 operation)
	1	Function call without additional DELAY
1	0	Turn OFF RF-field after operation completion (required for the last TELID242 operation)
	1	Leave the RF-field ON after operation completion
2	0	Read TELID242 ReadOnly-Code during function call
	1	Function call without additional reading of RO-Code

Result : returns 0 if successfull

Description : Command is reading the TELID242 sensor state.
 Please see hardware documentation for further parameter information.

int __stdcall APIENTRY __stdcall telid242_c_get_calibration (HANDLE m_HCom, int param, int *pCalibration, BYTE* pByteArray, BYTE *pCode) preliminary (Ver 0x10.0xC)

Parameter : *m_Hcom* - handle of port to be worked with
param - parameter byte for additional function control
 (see *telid242_c_get_status*)
pCalibration - address of buffer to store data in (7 integer values, value[0] contains length)
pByteArray - address of buffer to store raw data in (9 bytes, byte[0] contains length)
 Result : returns 0 if successfull
 Description : Command is reading the TELID242 sensor calibration data.
 Please see hardware documentation for further parameter information.

int __stdcall APIENTRY __stdcall telid242_c_get_temperature (HANDLE m_HCom, int param, int *pCalibration, double *pTemperature, BYTE *pByteArray, BYTE *pCode) preliminary (Ver 0x10.0xC)

Parameter : *m_Hcom* - handle of port to be worked with
param - parameter byte for additional function control
 (see *telid242_c_get_status*)
pCalibration - address of buffer containing calibration data (7 integer values, value[0] contains length)
pByteArray - address of buffer to store raw data in (3 bytes, byte[0] contains length)
**pTemperature* - pointer to double containing temperature data (in deg) after successful completion of command
 Result : returns 0 if successfull
 Description : Command is reading the TELID242 sensor temperature. Prior call of *telid242_c_get_calibration* is required.
 Please see hardware documentation for further parameter information.

int __stdcall APIENTRY __stdcall telid242_c_get_pressure (HANDLE m_HCom, int param, int *pCalibration, double TemperatureValue, double *pPressure, BYTE *pByteArray, BYTE *pCode) preliminary (Ver 0x10.0xC)

Parameter : *m_Hcom* - handle of port to be worked with
param - parameter byte for additional function control
 (see *telid242_c_get_status*)
pCalibration - address of buffer containing calibration data (7 integer values, value[0] contains length)
pByteArray - address of buffer to store raw data in (3 bytes, byte[0] contains length)
TemperatureValue – actual temperature value (in deg) determined using *telid242_c_get_temperature*
**pPressure* - pointer to double containing pressure data (in mbar) after successful completion of command
 Result : returns 0 if successfull
 Description : Command is reading the TELID242 sensor pressure. Prior call of *telid242_c_get_calibration* and *telid242_c_get_temperature* is required.
 Please see hardware documentation for further parameter information.

9. I-Code[®] UID TAG functions

This chapter describes functions available for communication with transponders based on the Philips I-Code[®] UID system.

BYTE __stdcall APIENTRY __stdcall icu_c_read_uid (HANDLE m_HCom, BYTE *pByteArray);

Parameter : m_HCom – handle of port with RFID interface
 *pByteArray – pointer to array of byte, which contains interface data after successfull completion of instruction, byte 0 contains length of data (5)

Result : success - 0
 error – see error table

Description : function reads UID from I-Code[®] UID transponder near RFID interface

BYTE __stdcall APIENTRY __stdcall icu_c_read_data (HANDLE m_HCom, INT32 from, INT32 length, BYTE *pByteArray, BYTE check_crc);

Parameter : m_HCom – handle of port with RFID interface
 from – address (0..11) of transponder to read data from
 length – length of data to read from transponder (1..12)
 check_crc - 0 – CRC check disabled for read operation (not recommended)
 - 1 – CRC16 check at byte[12..13] according product specification SL2 ICS11 Rev 3.0/2004 Jan 30
 - others – reserved for future development
 *pByteArray – pointer to array of byte, which contains interface data after successfull completion of instruction, byte 0 contains length of data

Result : success - 0
 error – see error table

Description : function reads partial data from I-Code[®] UID transponder near RFID interface

BYTE __stdcall APIENTRY __stdcall icu_c_write_data (HANDLE m_HCom, INT32 from, INT32 length, BYTE *pByteArray, BYTE check_crc);

Parameter : m_HCom – handle of port with RFID interface
 from – address (0..11) of transponder to write data to
 length – length of data to read from transponder (1..12)
 check_crc - 0 – CRC generation disabled for write operation (not recommended)
 - 1 – CRC16 generation at byte[12..13] according product specification SL2 ICS11 Rev 3.0/2004 Jan 30
 - others – reserved for future development
 *pByteArray – pointer to array of byte, which contains data to write to transponder

Result : success - 0
 error – see error table

Description : function writes partial data to I-Code[®] UID transponder near RFID interface
 (Attention: Because of the transponder functionality with external CRC generation It is recommended to write the whole data area (from=0, length=12) in one step to initialize the transponder, partial writing is supported, but not recommended)

BYTE __stdcall APIENTRY __stdcall icu_c_write_byte (HANDLE m_HCom,INT32 BlockAddress, BYTE *pByteArray);

Parameter	:	m_HCom – handle of port with RFID interface BlockAddress – address (0..13) of transponder to write data to *pByteArray – pointer to byte, which contains data to write to transponder
Result	:	success - 0 error – see error table
Description	:	function writes 1 byte to I-Code [®] UID transponder near RFID interface without CRC generation

10. I-Code[®] 1 TAG functions

This chapter describes functions available for communication with transponders based on the Philips RFID system I-Code[®] 1.

BYTE __stdcall APIENTRY __stdcall ic1_c_read_serial (HANDLE m_HCom, BYTE* pByteArray);

Parameter : m_HCom – handle of port with RFID interface
 *pByteArray – pointer to array of byte, which contains interface data after
 successfull completion of instruction, byte 0 contains length of data

Result : success - 0
 error – see error table

Description : function reads serial number from I-Code[®]1 transponder near RFID interface

BYTE __stdcall APIENTRY __stdcall ic1_c_read_block (HANDLE m_HCom, BYTE BlockAddress, BYTE* pByteArray);

Parameter : m_HCom – handle of port with RFID interface
 BlockAddress – address of transponder to read data from
 *pByteArray – pointer to array of byte, which contains interface data after
 successfull completion of instruction, byte 0 contains length of data

Result : success - 0
 error – see error table

Description : function reads block from I-Code[®]1 transponder near RFID interface

BYTE __stdcall APIENTRY __stdcall ic1_c_write_block (HANDLE m_HCom, BYTE BlockAddress, BYTE* pByteArray);

Parameter : m_HCom – handle of port with RFID interface
 BlockAddress – address of transponder to write data to
 *pByteArray – pointer to array of byte, which contains data to write to
 transponder

Result : success - 0
 error – see error table

Description : function writes block to I-Code[®]1 transponder near RFID interface

BYTE __stdcall APIENTRY __stdcall ic1_read_familycode (HANDLE m_HCom, INT32 *fc, INT32 *ai);

Parameter : m_HCom – handle of port with RFID interface
 fc – pointer to variable, which should contain family code of I-Code[®]1
 transponder
 ai – pointer to variable, which should contain application identifier of I-Code[®]1
 transponder

Result : success - 0
 error – see error table

Description : function reads FC and AI from I-Code[®]1 transponder near RFID interface

BYTE __stdcall APIENTRY __stdcall ic1_set_familycode (HANDLE m_HCom, INT32 fc, INT32 ai);

Parameter : m_HCom – handle of port with RFID interface
fc – new family code for I-Code®1 transponder
ai – new application identifier for I-Code®1 transponder
Result : success - 0
error – see error table
Description : function writes FC and AI to I-Code®1 transponder near RFID interface

BYTE __stdcall APIENTRY __stdcall ic1_read_access (HANDLE m_HCom, BYTE *pByteArray);

Parameter : m_HCom – handle of port with RFID interface
*pByteArray – pointer to array of byte, which contains data with access information
Result : success - 0
error – see error table
Description : function writes block to I-Code®1 transponder near RFID interface

BYTE __stdcall APIENTRY __stdcall ic1_set_access (HANDLE m_HCom, BYTE BlockAddress);

Parameter : m_HCom – handle of port with RFID interface
BlockAddress – address of transponder to lock
Result : success - 0
error – see error table
Description : function locks block of I-Code®1 transponder near RFID interface
(Att.: This command is irreversible !)

11. iID[®]-D TAG functions (preliminary)

BYTE __stdcall APIENTRY __stdcall iidd_c_read_ro_readonly (HANDLE m_HCom, BYTE *pByteArray);

BYTE __stdcall APIENTRY __stdcall iidd_c_read_ro_code (HANDLE m_HCom, BYTE *pByteArray);

BYTE __stdcall APIENTRY __stdcall iidd_c_write_ro_code (HANDLE m_HCom, BYTE *pByteArray);

BYTE __stdcall APIENTRY __stdcall iidd_c_read_block_pw (HANDLE m_HCom, INT32 BlockAddress, BYTE *pByteArray, BYTE *pPwArray);

BYTE __stdcall APIENTRY __stdcall iidd_c_read_block (HANDLE m_HCom, INT32 BlockAddress, BYTE *pByteArray);

BYTE __stdcall APIENTRY __stdcall iidd_c_send_read_protect_code (HANDLE m_HCom, BYTE *pPwArray);

BYTE __stdcall APIENTRY __stdcall iidd_c_prog_read_protect_code (HANDLE m_HCom, BYTE *pMasterCode, BYTE *pOldPw, BYTE *pPwArray);

BYTE __stdcall APIENTRY __stdcall iidd_c_write_block_pw (HANDLE m_HCom, INT32 BlockAddress, BYTE *pByteArray, BYTE *pPwArray);

BYTE __stdcall APIENTRY __stdcall iidd_c_write_block (HANDLE m_HCom, INT32 BlockAddress, BYTE *pByteArray);

BYTE __stdcall APIENTRY __stdcall iidd_c_send_write_protect_code (HANDLE m_HCom, BYTE *pPwArray);

BYTE __stdcall APIENTRY __stdcall iidd_c_prog_write_protect_code (HANDLE m_HCom, BYTE *pMasterCode, BYTE *pOldPw, BYTE *pPwArray);

BYTE __stdcall APIENTRY __stdcall iidd_c_get_temperature (HANDLE m_HCom, float *temperature, BYTE *pByteArray);

BYTE __stdcall APIENTRY __stdcall iidd_c_set_lockbit (HANDLE m_HCom, BYTE param);

12. LEGIC[®] Prime TAG functions (preliminary)

This chapter describes functions available for communication with transponders based on the LEGIC RFID transponders, type 'Prime'.

BYTE __stdcall APIENTRY __stdcall legic_c_read_uid (HANDLE m_HCom, BYTE* pType, BYTE* pByteArray);

Parameter	:	m_HCom – handle of port with RFID interface *pType – pointer to BYTE, which contains transponder information after successful completion of instruction *pByteArray – pointer to array of byte, which contains interface data after successful completion of instruction, byte 0 contains length of data
Result	:	success - 0 error – see error table
Description	:	function reads serial number from LEGIC [®] Prime transponder near RFID interface

BYTE __stdcall APIENTRY __stdcall legic_c_read_data (HANDLE m_HCom, BYTE param, BYTE* pType, BYTE* pByteArray); ¹

Parameter	:	m_HCom – handle of port with RFID interface param – BYTE parameter containing instruction-specific informations *pType – pointer to BYTE, which contains transponder information after successful completion of instruction *pByteArray – pointer to array of byte, which contains interface data after successful completion of instruction, byte 0 contains length of data
Result	:	success - 0 error – see error table
Description	:	function reads custom specific data from LEGIC [®] Prime transponder near RFID interface

¹ customized command – please ask MICROSENSYS depending further information

13. Mifare UltraLight™ TAG functions (preliminary)

This chapter describes functions available for communication with transponders based on the Mifare UltraLight system.

BYTE __stdcall APIENTRY __stdcall mifare_ul_c_read_uid (HANDLE m_HCom, BYTE *pByteArray)

Parameter : m_HCom – handle of port with RFID interface
 *pByteArray – pointer to array of byte, which contains interface data after successfull completion of instruction, byte 0 contains length of data, bytes 7..8 are not part of UID and contain manufacturer information

Result : success - 0
 error – see error table

Description : function reads UID from Mifare UltraLight™ transponder near RFID interface

BYTE __stdcall APIENTRY __stdcall mifare_ul_c_read_block_16 (HANDLE m_HCom, INT32 BlockAddress, BYTE param, BYTE *pByteArray)

Parameter : m_HCom – handle of port with RFID interface
 BlockAddress – address of transponder to read data from
 param : 0 – read/write using Mifare data offset
 1 – read/write starting from adress 0
 *pByteArray – pointer to array of byte, which contains interface data after successfull completion of instruction, byte 0 contains length of data

Result : success - 0
 error – see error table

Description : function reads block from Mifare UltraLight™ transponder near RFID interface

BYTE __stdcall APIENTRY __stdcall mifare_ul_c_write_block_4 (HANDLE m_HCom, INT32 BlockAddress, BYTE param, BYTE *pByteArray)

Parameter : m_HCom – handle of port with RFID interface
 BlockAddress – address of transponder to write data to
 param : 0 – read/write using Mifare data offset
 1 – read/write starting from adress 0
 *pByteArray – pointer to array of byte, which contains data to write to transponder

Result : success - 0
 error – see error table

Description : function writes block to Mifare UltraLight™ transponder near RFID interface

14. 125/134 kHz system TAG functions

Following described are some functions available for communication with 125kHz/134kHz transponders of several suppliers.

BYTE __stdcall APIENTRY __stdcall titan_c_read_block (HANDLE m_HCom, BYTE BlockAddress, BYTE* pByteArray);

Parameter : m_HCom – handle of port with RFID interface
 *pByteArray – pointer to array of byte, which contains interface data after successfull completion of instruction, byte 0 contains length of data

Result : success - 0
 error – see error table

Description : function reads serial number from TITAN transponder near RFID interface

BYTE __stdcall APIENTRY __stdcall titan_c_write_block (HANDLE m_HCom, BYTE BlockAddress, BYTE* pByteArray);

Parameter : m_HCom – handle of port with RFID interface
 BlockAddress – address of transponder to write data to
 *pByteArray – pointer to array of byte, which contains data to write to transponder

Result : success - 0
 error – see error table

Description : function writes block to TITAN transponder near RFID interface

BYTE __stdcall APIENTRY __stdcall unique_c_read_identifier (HANDLE m_HCom, BYTE* pByteArray);

Parameter : m_HCom – handle of port with RFID interface
 *pByteArray – pointer to array of byte, which contains interface data after successfull completion of instruction, byte 0 contains length of data

Result : success - 0
 error – see error table

Description : function reads Identifier from UNIQUE transponder near RFID interface

BYTE __stdcall APIENTRY __stdcall fdxb_c_read_identifier (HANDLE m_HCom, BYTE* pByteArray);

Parameter : m_HCom – handle of port with RFID interface
 *pByteArray – pointer to array of byte, which contains interface data after successfull completion of instruction, byte 0 contains length of data

Result : success - 0
 error – see error table

Description : function reads Identifier from FDXB -protocol based transponder near RFID interface

BYTE __stdcall APIENTRY __stdcall psk_1_c_read_identifier (HANDLE m_HCom, BYTE* pByteArray);

Parameter : m_HCom – handle of port with RFID interface
 *pByteArray – pointer to array of byte, which contains interface data after successfull completion of instruction, byte 0 contains length of data

Result : success - 0
 error – see error table

Description : function reads Identifier from special PSK transponder near RFID interface (please ask MICROSENSYS for more information)

BYTE __stdcall APIENTRY __stdcall psk_2_c_read_identifier (HANDLE m_HCom, BYTE *pByteArray);

Parameter	:	m_HCom – handle of port with RFID interface *pByteArray – pointer to array of byte, which contains interface data after successfull completion of instruction, byte 0 contains length of data
Result	:	success - 0 error – see error table
Description	:	function reads Identifier from special PSK transponder near RFID interface (please ask MICROSENSYS for more information)

15. Error - Codetable

State	Error
0x00	no error
0x08	identifiers do not match
0x11	range error
0x23	TAG-error
0x24	no TAG near antenna
0x26	OP-CODE unknown
0x28	protocol failure
0x29	unknown TAG instruction
0x2A	unknown TAG-error
0x2B	error writing to TAG
0x2C	error reading from TAG
0x2E	error control-reading TAG
0x2F	wrong data control-reading TAG (RAW)
0x30	error in CRC-Checksum
Additional Errors	
0x3F	Communication or port error
0x4F	unknown/not supported option detected
0xFF	general communication or driver error