iID® (mobile) driver engine

Product:	microsensys iID®3000 PRO RFID interfaces microsensys iID®4000 UHF RFID interfaces
Product Code:	**-**
Revision:	**0x10.0x5B**
Date:	2024-11-15

API - Definition for Windows® mobile and PC Windows®

iID® driver engine is the native driver dll supporting microsensys RFID interfaces. 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, Windows®Mobile, Windows®CE.Net, based on ARMV4 or X86 CPU.

This documentation describes the whole software API valid for many microsensys RFID interfaces supporting different functionalities.
A lot of applications are based on standard DOC RFID read/write operations, where we recommend usage of “iID® (mobile) driver functions”. Within more specific applications usage of TAG-specific functions could be required.

Drivername:	**iiddrv30_pro.dll**
Tested devices:	Windows® Embedded Handheld 6.5, Windows® Embedded Compact 7, Windows® Embedded Compact 2013
Supported host interface:	RS232 serial, Bluetooth SPP
Version:	0x10.0x5A
Release date:	2023-06-13
Tested devices:	MICROSOFT® Windows 7/64bit, MICROSOFT® Windows 8.1/32bit, MICROSOFT® Windows 10/64bit, MICROSOFT® Windows 11/64bit
Supported host interface:	RS232 serial, USB, Bluetooth SPP
Version:	0x10.0x5B
Release date:	2024-11-15
Charset:	Windows® ANSI charset

Content

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 official CD release
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)
2007-09-20	- added support for microsensys Pocket Reader iID®21x0 MPC	- added: iid2100_c_get_datacount, iid2100_c_get_dataset, iid2100_c_set_datasetnumber, iid2100_c_get_datacount	0x10.0xD	iiddrv30_pro.dll (only PC / WIN32)
2007-10-12	- extended LEGIC® Prime functionality	- added: legic_c_read_segment	0x10.0xE	iiddrv30_pro.dll (only PC / WIN32)
2007-10-30	- extended support for Inside contactless Pico-TAG	- added: picotag_c_read_uid picotag_c_write_block_8 picotag_c_read_block_8	0x10.0x10	iiddrv30_pro.dll
2008-03	- extended support TELID®242	- modified: telid242_c_get_status telid242_c_get_temperature telid242_c_get_pressure	0x10.0x11	iiddrv30_pro.dll
2008-04-08	- modification regarding my-D 2k SRF55V02P	- modified: c_get_transponder_parameters	0x10.0x12	iiddrv30_pro.dll official CD release
2008-04-21	- modification for 125kHz interfaces	- modified: c_set_interface_type	0x10.0x13	iiddrv30_pro.dll
2008-05-13	- modifications iID2110 functions, new application param 0x22	- modified iID2110 functions - added	0x10.0x14	iiddrv30_pro.dll
2008-10	- thread protection of serial communication functions - added ISO15693 functions	- modified communication framework - added: iso15693_c_get_multiple_block_security_status	0x10.0x16	iiddrv30_pro.dll official CD release
2009-04	- added iID-A functions	- added: c_iida_get_temperature	0x10.0x21	iiddrv30_pro.dll

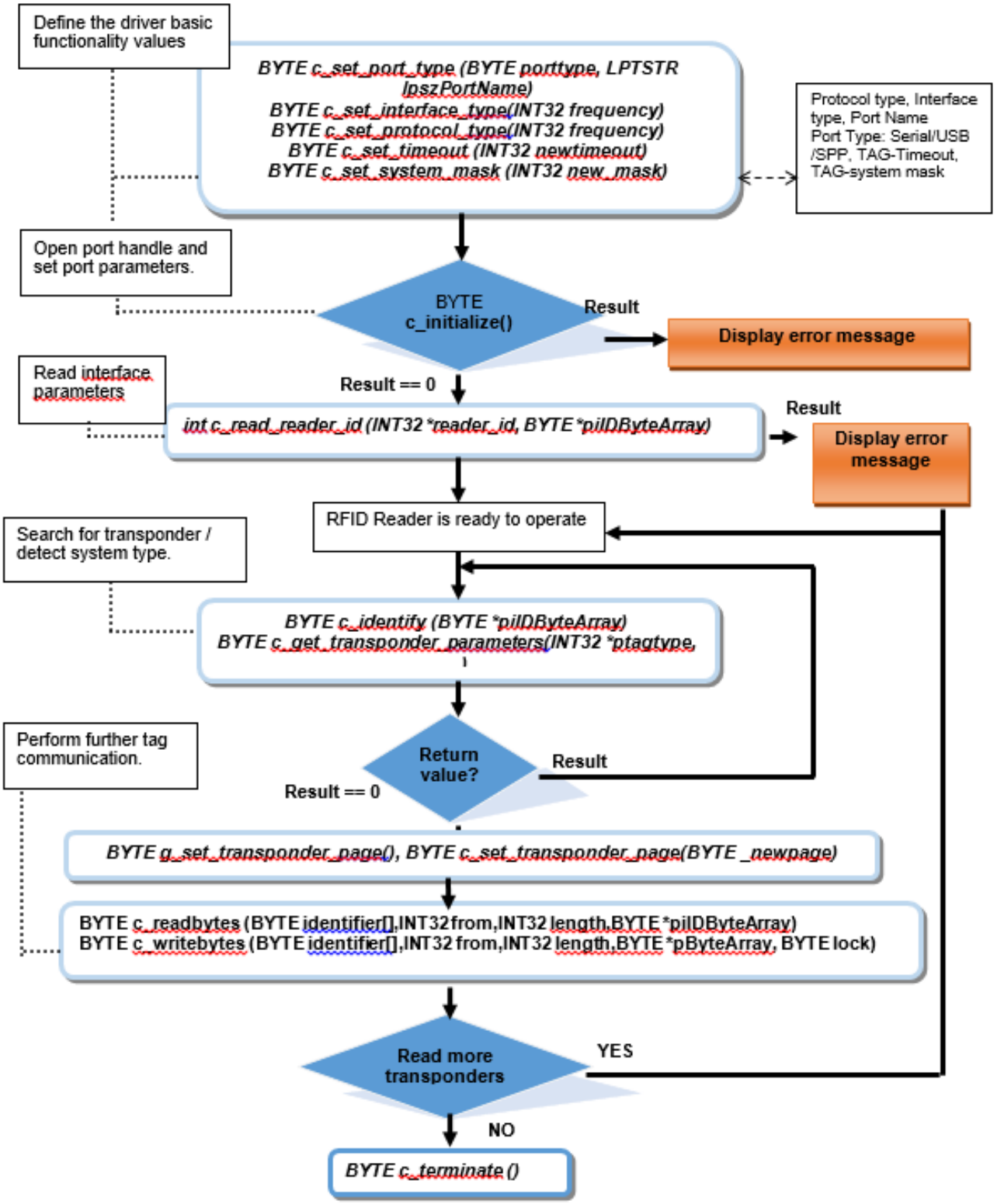
2009-08	- added iID-K functions (without security)	- added: iidk_c_read_uid iidk_c_read_linear iidk_c_write_linear	0x10.0x22	iiddrv30_pro.dll
2010-01	- added ISO15693 functions - added sensor function - improved timeout management	- added: telid243_c_get_data iso15693_myd_authenticate_a iso15693_myd_authenticate_b iso15693_myd_read_page iso15693_myd_write_page - resource optimized serial timeout	0x10.0x23	iiddrv30_pro.dll
2010-02	- added ISO15693 functions	- added: iso15693_myd_restricted_write_page iso15693_myd_write_and_reread_page iso15693_myd_restricted_write_and_reread_page	0x10.0x23	iiddrv30_pro.dll
2010-03		- modified for functionality in UM and IM: iso15693_myd_read_page iso15693_myd_write_page	0x10.0x23	iiddrv30_pro.dll
2010-05	- modified T-mode support		0x10.0x24	iiddrv30_pro.dll
2010-07	- added SAM emulation - added x64 release	- added: c_communicate_sam_data (described in additional documents)	0x10.0x26	iiddrv30_pro.dll official CD release
2012-01	- added iID interface protocol V4 - added page set/get functionality	- added: 868MHz interface type c_get_interface_type c_set_protocol_type c_get_protocol_type c_get_transponder_page c_set_transponder_page c_get_extended_status_information	0x10.0x31	Iiddrv30_pro.dll
2012-07	- Extended support of UHF functionality - added support for TELID(R)4xx / TELID(R)5xx sensor devices - protocol selection - page selection according ISO18000-6 and ISO18000-3M3 - extended status information - added LEGIC(TM) direct mode	- added c_set_page, c_get_page, c_get_extended_status_information - modified c_set_protocol_type, c_get_protocol_type,	010.0x40	Iiddrv30_pro.dll
2012-11	- extended support for LEGIC™ reader protocol within iID® contactless system - extended ISO15693 functionality	- modified - c_identify - c_readbytes - c_writebytes - added - iso15693_write_afi - iso15693_get_afi - iso15693_write_dsfid - iso15693_get_dsfid	010.0x41	Iiddrv30_pro.dll

2013-07	- extended ISO18000-6c functionality - support for I-Code SLI-X - added support for USB power up/down functionality	- added c_get_last_rssi - modified c_get_transponder_parameters c_identify - modified c_openinterface c_closeinterface	010.0x42	iiddrv30_pro.dll
2013-09	- support for IID®-F - improved direct USB communication	- modified USB communication functions - modified c_get_transponder_parameters c_identify	010.0x43	iiddrv30_pro.dll
2013-10	- extended ISO18000-6 (sensor) functionality	- modified c_get_temperature	010.0x44	iiddrv30_pro.dll
2013-12	- implementation of LEGIC file system support into iID driver functions	- modified c_set_system_mask c_set_baud_rate c_identify c_readbytes c_writebytes	0x10.0x46	iiddrv30_pro.dll
2014-01	- extended ISO18000-6 functionality	- implementation of iiD interface protocol V4 KILL command	0x10.0x46	iiddrv30_pro.dll
2014-04	- extended ISO18000-6 functionality - extended support iID interface protocol V4	- implementation UHF configuration commands - added LOCK support for iID-G - added LOCK support for inside contactless PICO-TAG - added interface protocol V4 functions - added MPC/offline functionality	0x10.0x48	iiddrv30_pro.dll
2014-06	c_transparent_hf	- added support for broken bits	0x10.0x4A	iiddrv30_pro.dll
2014-07	c_get_sensor	- extension for new sensor types	0x10.0x4B	iiddrv30_pro.dll
2014-09	c_set_RF_frequency_intprotv4	- added index parameter	0x10.0x4C	iiddrv30_pro.dll
2014-11	- modified/extended Legic direct functions - modified/extended UHF transparent functions		0x10.0x4D	iiddrv30_pro.dll
2015-04	legic_c_read_data c_get_transponder_parameters extended ISO14443 functionality	- modified in Legic direct functions - extended tag type detection for ISO14443 - added: iso14443a_read_transparent iso14443a_write_transparent ntag_password_auth	0x10.0x4E	iiddrv30_pro.dll
2015-07	Added UHF AFI specific functions Added data encode/decode functions	Added: c_set_EPC_extended_parameter, c_get_EPC_extended_parameter, c_set_AFI_mask, c_get_AFI_mask, c_uii_decode, c_uii_encode	0x10.0x4F	iiddrv30_pro.dll
2015-07	Parameter modification	Modification: c_uii_decode, c_uii_encode c_get_AFI_mask	0x10.0x50	iiddrv30_pro.dll
2015-08	Logic modification	Modification: legic_direct_reader_id – implementation of 96bit LEGIC reader ID legic_c_read_data		
2015-08	Release	Modification of test platforms (see tested devices)	0x10.0x51	iiddrv30_pro.dll

2016-01	Functional modifications	Legic SM4x00 checksum calculation Internal BT interface access: sequence number creation in c_readfile, c_writefile legic_c_read_data(param 0x20): output data modification iidl_c_get_sensor_data: added T242.02 mobile releases: modified power_on behaviour	0x10.0x52	iiddrv30_pro.dll
	Added functions	c_reader_set_operation_mode c_reader_get_operation_mode c_reader_set_sleep	0x10.0x52	iiddrv30_pro.dll
2016-04	Functional modifications	Internal BT interface access: sequence number creation modified C_openinterface: USB access optimized by scanning for device name	0x10.0x53	iiddrv30_pro.dll
	Added functions	Keller pressure sensors: added offset & amplification	0x10.0x53	iiddrv30_pro.dll
2016-06	Functional modifications	C_set_protocol_type, c_set_port_type: added support for CASIO integrated NFC interfaces Iso14443a transparent functions: extended chip type support c_set_selected_antenna_intprotv4: added support for INDUSTRYeight	0x10.0x54	iiddrv30_pro.dll
	Added functions	Added LEGIC_direct functions *please ask microsensys for detailed information	0x10.0x54	iiddrv30_pro.dll
2016-09	Functional modifications	legic_c_read_data(param 0x20): output data modification	0x10.0x55	iiddrv30_pro.dll
	Added functions	c_set_AutoOffUhf_intprotv4	0x10.0x55	iiddrv30_pro.dll
2016-12	Functional modifications	ISO18000-6C read/write functions return “UNKNOWN_OPTION” when from/length are odd	0x10.0x56	iiddrv30_pro.dll pre-release
	Added function	iso18000_6_c_read_words_secure iso18000_6_c_write_words_secure	0x10.0x56	iiddrv30_pro.dll pre-release
2017-01	Release CD-ROM 2017-01	iso18000_6_c_read_words_secure: modification for final release	0x10.0x56	iiddrv30_pro.dll
2017-05	Release CD-ROM 2017-01 & WEB	Removed blank page 59 Modified port type for CASIO integrated NFC interface	0x10.0x56	iiddrv30_pro.dll
2017-06	Release CD-ROM 2017-02 & WEB	Added UHF functions • iso18000_6_c_read_epc_rssi_gen2 • iso18000_6_c_lock_secure • iso18000_6_c_write_epc_gen2 Added LEGIC reader functions • legic_c_search_segment Added reader function for customer use • c_reader_prog_custom_memory • c_reader_read_custom_memory	0x10.0x57	iiddrv30_pro.dll
2019-10	Added functionality	• Bug fixing • Added support for iID-M2 • Added support for divers sensors TELID211.03, …	0x10.0x58	iiddrv30_pro.dll
2020-03	Added functionality	Added commands for improved ISO15693 read/write communication speed (reader firmware support required) Added ISO15693 functions • iso15693_read_blocks • iso15693_write_blocks • iso15693_inventory_anticollision	0x10.0x59	iiddrv30_pro.dll

		Added chip support: EM echo		
2023-06	Added functionality	Added commands for LDR02	0x10.0x5A	iiddrv30_pro.dll
2024-11	Solved issues	Added checks to avoid buffer overflows for LDR02 implementation	0x10.0x5B	iiddrv30_pro.dll

Flow diagram – How to use iID® driver engine



Common functions

Communication port handling

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)
- 5 – EP-10 integrated LEGIC reader interface*
- 6 – CASIO integrated NFC reader interface*

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
- 0x364/868 – 868MHz RFID interface mode

Result : success - 0
error – 0xFF
Description : function sets port parameters for c_initialize and c_openinterface and other iID® driver functions.

INT32 __stdcall APIENTRY __stdcall c_get_interface_type();

Parameter : -
Result : frequency
- 0x7D/125 – 125/134kHz RFID interface mode
- 0x54C/1356 – 13.56MHz RFID interface mode
- 0x364/868 – 868MHz/915MHz RFID interface mode

Description : function gets the frequency set in the driver with c_set_interface_type(INT32 frequency).

INT32 __stdcall APIENTRY __stdcall c_get_protocol_type();

Parameter : -
Result : returns actual protocol type
- 0x2 – iID 2000 protocol
- 0x3 – iID 3000 protocol
- 0x4 – iID interface protocol V4
- 0x1001/4097 – CAEN reader protocol
- 0x1002/4098 – LEGIC™ direct protocol
- 0x1003/4099 – CASIO integrated NFC reader protocol*

Description : Command gets the actual protocol which is used within iID driver functions

** available on request*

BYTE __stdcall APIENTRY __stdcall c_set_protocol_type(INT32 _iProtocol);

Parameter	:	_iProtocol	- 0x2 – iID 2000 protocol - 0x3 – iID 3000 protocol - 0x4 – iID interface protocol V4 - 0x1001/4097 – CAEN protocol - 0x1002/4098 – LEGIC™ direct protocol - 0x1003/4099 – CASIO integrated NFC reader protocol*
Result	:	returns 0, if successful or 0x4F for unsupported options	
Description	:	Command sets the protocol to use for within iID driver functions *available only on selected CASIO industrial mobile handhelds with integrated NFC reader	

iID® 3000 PRO protocol functions

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

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 code table
Description	:	function reads ID-number and parameters from RFID interface with iID®3000 protocol frame

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 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 code 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 code table
Description	:	function sets parameters for RFID interface

int __stdcall APIENTRY __stdcall c_reader_set_auto_off_iid3000 (HANDLE m_HCom, BYTE param);

Parameter	:	m_HCom – handle of port with RFID interface param – antenna off information 0 – switch off antenna immediately after RF operation 1..0xFE – switch off antenna after x X 10msec after RF operation 0xFF – leave antenna on after RF operation
Result	:	success - 0 error – see error code table
Description	:	function sets antenna switch OFF parameter 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 code 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 code table
Description	:	function switches OFF LED of RFID interface This command is only available on devices with LED support.

BYTE __stdcall APIENTRY __stdcall c_reader_set_service_iid3000 (HANDLE m_HCom, BYTE param)

Parameter	:	m_HCom - handle of port, where RFID interface is connected param - byte with control information for RFID interface
Result	:	success - 0 error – see error code table
Description	:	function sets different RFID interface options

iID® interface protocol V4 functions

This chapter describes general control functions of iID® interface protocol V4 compatible readers.

BYTE __stdcall APIENTRY __stdcall c_read_reader_id_intprotv4(HANDLE m_HCom, INT32 *_uReaderId, BYTE *_bData)

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 code table
Description : function reads ID-number and parameters from RFID interface with iID®3000 protocol frame

INT32 __stdcall APIENTRY __stdcall c_get_extended_status_information();

Result : returns extended status information
Description : Command to get the extended status information from c_identify, c_readbytes and c_writebytes when using iID interface protocol V4
Parameter : -
Result : extended error code (see hardware documentation)
Description : Command returns extended error information of prior RF operation, if provided by RFID interface. Extended status information will be lost when calling the next interface protocol V4 function.

BYTE __stdcall APIENTRY __stdcall c_reader_set_operation_mode(HANDLE m_HCom, UINT _bDriverParam, BYTE _bParam, bool _performReset)

Parameter : m_HCom – handle of port with RFID interface
_bDriverParam – internal parameter, should be set to 0
_bParam - byte with control information for RFID interface

Value (hex)	Description
0x00	Switch RFID interface to DOC mode
0x01	Switch RFID interface to SPC mode

_performReset - byte with control information for RFID interface

Value (boolean)	Description
False	Do not restart RFID interface
True	Restart RFID interface

Result : success - 0
error – see error code table
Description : function changes RFID interface operation mode (DOC/SPC)

BYTE __stdcall APIENTRY __stdcall c_reader_get_operation_mode(HANDLE m_HCom, UINT _bDriverParam, BYTE *_bParam)

Parameter : m_HCom – handle of port with RFID interface
_bDriverParam – internal parameter, should be set to 0
_bParam - byte contains current RFID interface operation mode after successful command execution

Value (hex)	Description
0x00	RFID interface operates in DOC mode
0x01	RFID interface operates in SPC mode

Result : success - 0
error – see error code table
Description : function returns RFID interface operation mode (DOC/SPC)

BYTE __stdcall APIENTRY __stdcall c_stop_script_intprotv4(HANDLE m_HCom, BYTE _StdParam, UINT32 _driverParam)

Description : internal function, not for general usage

BYTE __stdcall APIENTRY __stdcall c_set_buzzer_intprotv4(HANDLE m_HCom, BYTE _StdParam, UINT32 _driverParam, BYTE _bState, BYTE _bDelay)

Parameter : m_HCom - handle of port, where RFID interface is connected
_StdParam - byte with control information for RFID interface, should be set 0
_driverParam – internal parameter, should be set to 0
_bState – 0 = disable buzzer, 1 = enable buzzer
_bDelay – buzzer time 1...255 x 10msec
Result : success - 0
error – see error code table
Description : function activates/deactivates buzzer of compatible RFID interface.

BYTE __stdcall APIENTRY __stdcall c_set_display_intprotv4(HANDLE m_HCom, BYTE _StdParam, UINT32 _driverParam, BYTE _bType, BYTE _bFormat, BYTE _bParamsLength, BYTE *_bParams)

Parameter : m_HCom - handle of port, where RFID interface is connected
_StdParam - byte with control information for RFID interface, should be set 0
_driverParam – internal parameter, should be set to 0
_bType – parameter configuring display message type

Data content (hex)	Description
0x00	Clear display
0x03	Show icon, see _bParams
0x04	Show Text data, see _bParams

_bFormat – parameter configuring display message format

Data content (binary)	Description
xxxx xx00	Hexadecimal value, large font
xxxx xx10	Hexadecimal value, small font
xxxx xx01	ASCII value, large font
xxxx xx11	ASCII value, small font

_ bParamsLength – contains length of _bParams array
_bParams – pointer to array of byte containing display data

Data content _bType	Content _bParams
0x00	No content
0x03	0x00 – show OKAY icon 0x01 - show FAIL icon 0x02 – show WORKING icon
0x04	Array of bytes/characters to display

Result : success - 0
error – see error code table
Description : function shows data on display of compatible RFID interface.

BYTE __stdcall APIENTRY __stdcall c_set_bluetooth_intprotv4(HANDLE m_HCom, BYTE _StdParam, UINT32 _driverParam, BYTE _bParam)

Parameter : m_HCom - handle of port, where RFID interface is connected
_StdParam - byte with control information for RFID interface, should be set 0
_driverParam – internal parameter, should be set to 0
_bParam – 0x00 = turn off, 0x01 = turn on
Result : success - 0
error – see error code table
Description : function activates/deactivates Bluetooth™ wireless functionality of compatible RFID interface.

BYTE __stdcall APIENTRY __stdcall c_get_batstate_intprotv4(HANDLE m_HCom, BYTE _StdParam, UINT32 _driverParam, BYTE *_bSensorType, UINT32 *_uMeasurement, UINT32 *_uReference, BYTE *_bData)

Description : internal function, not for general usage

BYTE __stdcall APIENTRY __stdcall c_set_DateTime_intprotv4(HANDLE m_HCom, BYTE _StdParam, UINT32 _driverParam, BYTE _bYear2000, BYTE _bMonth, BYTE _bDay, BYTE _bWDay, BYTE _bHour, BYTE _bMin, BYTE _bSec, int _iUTCoffsetMinutes)

Parameter : m_HCom - handle of port, where RFID interface is connected
_StdParam - byte with control information for RFID interface, should be set 0
_driverParam – internal parameter, should be set to 0
_bYear2000, _bMonth, … - values containing new device time
_iUTCoffsetMinutes – positive/negative UTC offset in minutes
Result : success - 0
error – see error code table
Description : function sets device time of supported RFID interfaces.

BYTE __stdcall APIENTRY __stdcall c_get_DateTime_intprotv4(HANDLE m_HCom, BYTE _StdParam, UINT32 _driverParam, BYTE *_bYear2000, BYTE *_bMonth, BYTE *_bDay, BYTE *_bWDay, BYTE *_bHour, BYTE *_bMin, BYTE *_bSec, int *_iUTCoffsetMinutes)

Parameter : m_HCom - handle of port, where RFID interface is connected
_StdParam - byte with control information for RFID interface, should be set 0
_driverParam – internal parameter, should be set to 0
_bYear2000, _bMonth, … - values containing current device time*
_iUTCoffsetMinutes – value containing UTC offset in minutes
*after successful completion of command
Result : success - 0
error – see error code table
Description : function gets device time of supported RFID interfaces.

BYTE __stdcall APIENTRY __stdcall c_get_hardware_information_intprotv4(HANDLE m_HCom, BYTE _StdParam, UINT32 _driverParam, BYTE *_bDevice, BYTE *_bHW, BYTE *_bFW, BYTE *_bProtocol, BYTE *_bNumASIC, BYTE *_bRFData, BYTE *_bData)

Description : internal function, not for general usage

BYTE __stdcall APIENTRY __stdcall c_soft_reset_intprotv4(HANDLE m_HCom, BYTE _StdParam, UINT32 _driverParam)

Description : internal function, not for general usage

BYTE __stdcall APIENTRY __stdcall c_get_hardware_information_intprotv4(HANDLE m_HCom, BYTE _StdParam, UINT32 _driverParam, BYTE *_bDevice, BYTE *_bHW, BYTE *_bFW, BYTE *_bProtocol, BYTE *_bNumASIC, BYTE *_bRFData, BYTE *_bData)

Description : internal function, not for general usage

BYTE __stdcall APIENTRY __stdcall c_set_output_intprotv4(HANDLE m_HCom, BYTE _StdParam, UINT32 _driverParam, BYTE _bOutMask, BYTE _bDelay)

Description : internal function, not for general usage

BYTE __stdcall APIENTRY __stdcall c_set_led_intprotv4(HANDLE m_HCom, BYTE _StdParam, UINT32 _driverParam, BYTE _bLedMask, BYTE _bDelay)

Description : internal function, not for general usage

UHF specific RFID interface functions

BYTE __stdcall APIENTRY __stdcall c_set_power_intprotv4(HANDLE m_HCom, BYTE _StdParam, UINT32 _driverParam, BYTE _bBoost1, BYTE _bBoost2)

Parameter : m_HCom - handle of port, where RFID interface is connected
_StdParam - byte with control information for RFID interface, should be set 0
_driverParam – internal parameter, should be set to 0
_bBoost1 – internal, should be set to 0x03
_bBoost2 – amplifier factor 1..3
Result : success - 0
error – see error code table
Description : function sets current RF power value of RFID interface.

BYTE __stdcall APIENTRY __stdcall c_set_RF_state_intprotv4(HANDLE m_HCom, BYTE _StdParam, UINT32 _driverParam, BYTE _bRFState)

Parameter : m_HCom - handle of port, where RFID interface is connected
_StdParam - byte with control information for RFID interface, should be set 0
_driverParam – internal parameter, should be set to 0
_bRFState – 0 = disable RF power, 1 = enable RF power
Result : success - 0
error – see error code table
Description : function activates/deactivates RF output of RFID interface.

BYTE __stdcall APIENTRY __stdcall c_set_selected_antenna_intprotv4(HANDLE m_HCom, BYTE _StdParam, UINT32 _driverParam, BYTE _bAntennaBitMask)

Parameter : m_HCom - handle of port, where RFID interface is connected
_StdParam - byte with control information for RFID interface, should be set 0
_driverParam – internal parameter, should be set to 0
_bAntennaBitMask – bitmask for antennas to be activated, multiple bits can be selected

Bit	Value	Description
X (0..7)	0	Antenna x (0..7) de-activated during scan
	1	Antenna x (0..7) activated during scan

Result : success - 0
error – see error code table
Description : function sets value of selected antenna for RF operations

BYTE __stdcall APIENTRY __stdcall c_set_AutoOffUhf_intprotv4(HANDLE m_HCom, BYTE _StdParam, UINT32 _driverParam, BYTE _bParam)

Parameter	:	m_HCom – handle of port with RFID interface UINT32 _uDriverParam – reserved, should be 0xFF UINT32 _uStdParam – hardware parameter, see hardware documentation BYTE *_bParam –byte, which contains antenna auto off time factor 10msec
Result	:	success - 0 error – see error code table
Description	:	function defines RFID interface automatic antenna OFF time (per 10msec)

BYTE __stdcall APIENTRY __stdcall c_get_last_RSSI(HANDLE m_HCom, BYTE *_bLastRSSIValue)

Parameter	:	m_HCom – handle of port with RFID interface BYTE *_bLastRSSIValue – pointer to byte containing RSSI value of previous ISO18000-6C operation
Result	:	success - 0 error – see error code table
Description	:	function returns RSSI value of previous ISO18000-6C operation from RFID interface

BYTE __stdcall APIENTRY __stdcall c_get_Q_intprotv4(HANDLE m_HCom, BYTE _StdParam, UINT32 _driverParam, BYTE *_bQ)

Parameter	:	m_HCom - handle of port, where RFID interface is connected _StdParam - byte with control information for RFID interface, should be set 0 _driverParam – internal parameter, should be set to 0 _bQ – pointer to byte containing current Q value after successful completion of command
Result	:	success - 0 error – see error code table
Description	:	function gets current Q value from RFID interface

BYTE __stdcall APIENTRY __stdcall c_set_Q_intprotv4(HANDLE m_HCom, BYTE _StdParam, UINT32 _driverParam, BYTE _bQ)

Parameter	:	m_HCom - handle of port, where RFID interface is connected _StdParam - byte with control information for RFID interface, should be set 0 _driverParam – internal parameter, should be set to 0 _bQ –byte containing new Q value
Result	:	success - 0 error – see error code table
Description	:	function sets current Q value of RFID interface

BYTE __stdcall APIENTRY __stdcall c_get_AutoHop_intprotv4(HANDLE m_HCom, BYTE _StdParam, UINT32 _driverParam, BYTE *_bAutoHopState)
BYTE __stdcall APIENTRY __stdcall c_set_AutoHop_intprotv4(HANDLE m_HCom, BYTE _StdParam, UINT32 _driverParam, BYTE _bAutoHopState)

Description : internal functions, not for general usage

BYTE __stdcall APIENTRY __stdcall c_get_HoppingStep_intprotv4(HANDLE m_HCom, BYTE _StdParam, UINT32 _driverParam, BYTE *_bHopStep)
BYTE __stdcall APIENTRY __stdcall c_set_HoppingStep_intprotv4(HANDLE m_HCom, BYTE _StdParam, UINT32 _driverParam, BYTE _bHopStep)

Description : internal functions, not for general usage

BYTE __stdcall APIENTRY __stdcall c_set_RF_frequency_intprotv4(HANDLE m_HCom, BYTE _StdParam, UINT32 _driverParam, INT32 _freqKHz)

Description : internal function, not for general usage

Data encoding/decoding functions

Following chapter keeps reader and transponder independent data encoding functions according ISO and/or EPC global standards.

BYTE __stdcall APIENTRY __stdcall c_uii_decode(UINT32 _driverParam, UINT32 _convertType, BYTE* _toDecode, int _from, int _length, BYTE* _decoded, int* _decodedLength);

Parameter : _driverParam – reserved, should be zero
_convertType – keeps conversion rule from _toDecode to _decoded

Value	Description
0	reserved
1	Decodes ISO17367 6bit reader response to 8bit byte values (_from and _length have to be filled properly)

*_toDecode – pointer to array of byte, which contains source data
from – byte start index for conversion of source data
length – byte length for conversion of source data
*_decoded – pointer to array of byte, which returns converted data
*_decodedLength – pointer to int value, which returns length of converted data

Result : success - 0
error – see error code table

Description : function converts source data array to target array using conversion rule defined in _convertType. . *This function is available up from driver version 0x10.0x4F.*

BYTE __stdcall APIENTRY __stdcall c_uii_encode(UINT32 _driverParam, UINT32 _convertType, BYTE* _toEncode, int _from, int _length, BYTE* _encoded, int* _encodedLength);

Parameter : _driverParam – reserved, should be zero
_convertType – keeps conversion rule from _toDecode to _decoded

Value	Description
0	reserved
1	Encodes 8 bit byte values to ISO17367 6bit data string (_from and _length have to be filled properly)

*_toEncode – pointer to array of byte, which contains source data
from – byte start index for conversion of source data
length – byte length for conversion of source data
*_encoded – pointer to array of byte, which returns converted data
*_encodedLength – pointer to int value, which returns length of converted data

Result : success - 0
error – see error code table

Description : function converts source data array to target array using conversion rule defined in _convertType. . *This function is available up from driver version 0x10.0x4F.*

iID® (mobile) driver functions

The following chapter describes 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.

iID® driver functions encapsulate reader-, frequency,- and tag-specific functions to abstract functions. If access to TAG-specific functions is required, you may need to use specific functions instead or additional to iID® driver functions.

BYTE __stdcall APIENTRY __stdcall c_initialize();

Parameter :
Result : returns 0 if successful, or error-code (see error code 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 successfuly closed or error-code (see error code 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 *pilDByteArray);

Parameter : *m_Hcom* - handle of 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 code 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 independent from protocol type.

BYTE __stdcall APIENTRY __stdcall c_reader_prog_custom_memory (HANDLE m_HCom, UINT32 _uDriverParam, UINT32 _uStdParam, BYTE *_pbData, BYTE _bDataLength);

Parameter : *m_Hcom* - handle of port to be worked with
_driverParam – reserved, should be zero
_StdParam – reserved, should be zero
*pbData – pointer to array of bytes containing data to program to non-volatile reader memory
_bDataLength – number of bytes to progran
Result : returns 0 if successful, or error-code (see error code table)
Description : Command programs data (max. 64 bytes) to non-volatile reader memory. This memory is available for customer use (only supported readers).

BYTE __stdcall APIENTRY __stdcall c_reader_read_custom_memory (HANDLE m_HCom, UINT32 _uDriverParam, UINT32 _uStdParam, BYTE *_pbByteArray);

Parameter	:	*m_Hcom* - handle of port to be worked with _driverParam – reserved, should be zero _StdParam – reserved, should be zero *pByteArray – pointer to array of bytes containing data, returns (byte 0 -> length of data, byte 1..n -> data)
Result	:	returns 0 if successful, or error-code (see error code table)
Description	:	Command is reading data from non-volatile reader memory, which was stored before using c_reader_prog_custom_memory.

BYTE __stdcall APIENTRY __stdcall c_identify (BYTE *pilDByteArray);

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 code 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 *pilDByteArray, 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 code 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 *pilDByteArray); *(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. *Up from driver version 0x10.0x4E a call of c_get_transponder_parameters for HF/ISO14443 compatible tags may cause some additional tag operations for type detection.*

BYTE __stdcall APIENTRY __stdcall c_set_transponder_page(UINT32 _page);

Parameter	:	_page - int32 containing page to read/write information
Result	:	returns 0 if successful
Description	:	Command sets page index to read/write for following c_readbytes / c_writebytes call

UINT32 __stdcall APIENTRY __stdcall c_get_transponder_page();

Parameter	:	-
Result	:	int32 containing current page index
Description	:	Command gets active page index to read/write for following c_readbytes / c_writebytes call

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_Identify() for more details)
Result	:	returns 0 if successful, or error-code (see error code 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]) pIdentifier - 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 code 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.

INT32 __stdcall APIENTRY __stdcall c_get_extended_status_information();

Please see chapter iID® interface protocol V4 functions.

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

For full list of supported transponder systems see chapters “Transponder Systems” and “Transponder chip types”.

0x1 – ISO 15693 /125kHz UNIQUE /ISO18000-6C transponders enabled/disabled and/or
0x2 – iID®-L /125kHz TITAN transponders enabled/disabled
….
0x40000 – LEGIC™ file system enabled/disabled (note 1: requires protocol type LEGIC™ direct (0x1002), note 2: use this mask in combination with LEGIC™ file system enabled chip types, *example: 0x40001: LEGIC™ file system on ISO15693 chip*) This functionality is available up from driver version 0x01.0x46.

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

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.

HF transponder chip specific driver functions

HF - ISO15693 standard 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 successful completion of instruction, byte 0 contains length of data
Result	:	success - 0 error – see error code 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 successful 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 code table
Description	:	function reads UID from ISO15693 transponder near RFID interface in Anticollison 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 successful completion of instruction, byte 0 contains length of data
Result	:	success - 0 error – see error code 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 successful completion of instruction, byte 0 contains length of data *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 code table
Description	:	function reads block from ISO15693 transponder near RFID interface in anticollision mode

BYTE __stdcall APIENTRY __stdcall iso15693_read_blocks (HANDLE m_HCom, UINT32 _uDriverParam, UINT32 _uStdParam, BYTE _bFlags, BYTE _bBlockSize, BYTE _bBlockNum, BYTE *pIdentifier, BYTE _bNumBytes, BYTE *pByteArray);

Parameter	:	m_HCom – handle of port with RFID interface _uDriverParam – reserved, should be 0xFF _uStdParam – reserved, should be 0xFF _bFlags – selects command parameters. 0 → Use ISO15693 “Read single block” command 1 → Use ISO15693 “Read multiple blocks” command _bBlockSize – transponder memory block size _bBlockNum – first transponder memory block to read *pIdentifier – pointer to array of byte, which contains identifier of transponder to select for read/write operation _bNumBytes – number of bytes to read from transponder (must be multiple of _bBlockSize) *pByteArray – pointer to array of byte, which contains data after successful completion of instruction, byte 0 contains length of data
Result	:	success - 0 error – see error code table
Description	:	function reads one or more blocks from ISO15693 transponder near RFID interface **This function is available up from driver version 0x10.0x59 for interfaces supporting “ISO15693 Read/Write Blocks” command.**

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

BYTE __stdcall APIENTRY __stdcall iso15693_write_blocks (HANDLE m_HCom, UINT32 _uDriverParam, UINT32 _uStdParam, BYTE _bFlags, BYTE _bBlockSize, BYTE _bBlockNum, BYTE *pIdentifier, BYTE _bNumBytes, BYTE *pByteArray);

Parameter	:	m_HCom – handle of port with RFID interface _uDriverParam – reserved, should be 0xFF _uStdParam – reserved, should be 0xFF _bFlags – selects command parameters. 0 → Use ISO15693 “Write single block” command 1 → Use ISO15693 “Write multiple blocks” command _bBlockSize – transponder memory block size _bBlockNum – first transponder memory block to write *pIdentifier – pointer to array of byte, which contains identifier of transponder to select for read/write operation _bNumBytes – number of bytes to write to transponder (must be multiple of _bBlockSize) *pByteArray – pointer to array of byte, which contains data to write to transponder
Result	:	success - 0 error – see error code table
Description	:	function writes one or more blocks to ISO15693 transponder near RFID interface **This function is available up from driver version 0x10.0x59 for interfaces supporting “ISO15693 Read/Write Blocks” command.**

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

BYTE __stdcall APIENTRY __stdcall iso15693_get_multiple_block_security_information (HANDLE m_HCom, BYTE *pSelectedUID, BYTE ISO_COM_Ctrl, BYTE REQ_Byte, BYTE BlockAddress, BYTE NrOfBlocks, BYTE *pByteArray);

Parameter	:	m_HCom – handle of port with RFID interface *pSelectedUID – has to be set to null / nil for common usage ISO_COM_Ctrl – has to be set to 0 for common usage REQ_Byte – has to be set to 2 for common usage BlockAddress – start address of transponder to get lock information NrOfBlocks – nr of blocks – 1 of transponder to get lock information (0 returns block security information of 1 block as defined in ISO15693 *pByteArray – pointer to array of byte, which contains interface data after successful completion of instruction (see ISO15693), byte 0 contains length of data (normally NrOfBlocks + 1 bytes)
Result	:	success - 0 error – see error table
Description	:	supported chips should return the block security status, when receiving this command (ISO 15693 command code 0x2C). **This function is available up from driver version 0x10.0x22.**

BYTE __stdcall APIENTRY __stdcall iso15693_write_AFI(HANDLE m_HCom, BYTE *pSelectedUID, BYTE AFI, BYTE *pByteArray)

Parameter	:	m_HCom – handle of port with RFID interface *pSelectedUID – has to be set to null / nil for common usage AFI – new application family identifier to write *pByteArray – pointer to array of byte, which contains data, for further information see ISO15693
Result	:	success - 0 error – see error table or ISO15693
Description	:	ISO15693 specific function to support AFI usage. **This function is available up from driver version 0x10.0x41 for interfaces supporting transparent ISO15693 functionality.**

BYTE __stdcall APIENTRY __stdcall iso15693_get_AFI(HANDLE m_HCom, BYTE *pSelectedUID, BYTE *AFI, BYTE *pByteArray)

Parameter	:	m_HCom – handle of port with RFID interface *pSelectedUID – has to be set to null / nil for common usage *AFI – pointer to byte containing application family identifier after successful completion of command *pByteArray – pointer to array of byte, which contains data, for further information see ISO15693
Result	:	success - 0 error – see error table or ISO15693
Description	:	ISO15693 specific function to support AFI usage. **This function is available up from driver version 0x10.0x41 for interfaces supporting transparent ISO15693 functionality.**

BYTE __stdcall APIENTRY __stdcall iso15693_write_DSFID(HANDLE m_HCom, BYTE *pSelectedUID, BYTE DSFID, BYTE *pByteArray)

Parameter	:	m_HCom – handle of port with RFID interface *pSelectedUID – has to be set to null / nil for common usage DSFID – new DSFID to write *pByteArray – pointer to array of byte, which contains data, for further information see ISO15693
Result	:	success - 0 error – see error table or ISO15693
Description	:	ISO15693 specific function to support AFI usage. **This function is available up from driver version 0x10.0x41 for interfaces supporting transparent ISO15693 functionality.**

BYTE __stdcall APIENTRY __stdcall iso15693_get_DSFID(HANDLE m_HCom, BYTE *pSelectedUID, BYTE *DSFID, BYTE *pByteArray)

Parameter	:	m_HCom – handle of port with RFID interface *pSelectedUID – has to be set to null / nil for common usage *DSFID – pointer to byte containing DSFID after successful completion of command *pByteArray – pointer to array of byte, which contains data, for further information see ISO15693
Result	:	success - 0 error – see error table or ISO15693
Description	:	ISO15693 specific function to support AFI usage. **This function is available up from driver version 0x10.0x41 for interfaces supporting transparent ISO15693 functionality.**

BYTE __stdcall APIENTRY __stdcall c_transparent_HF(HANDLE m_HCom, BYTE _bCtl, BYTE _bModulation, BYTE _bTXChecksum, BYTE _bRXCheck, BYTE _bWaitForRec, BYTE _bRecTimeout, BYTE _bTXBrokenBits, BYTE _bRXBrokenBits, BYTE _bTransparentLength, BYTE *_bTransparent, BYTE *_pByteArray)

Description	:	Function to handle transparent RF data communication using ISO14443/ISO15693 transmission protocols.

This function is available up from driver version 0x10.0x4A for interfaces supporting transparent ISO14443A,B/ISO15693 functionality. Please contact microsensys for further support about transparent reader functionality.

BYTE __stdcall APIENTRY __stdcall iso15693_inventory_anticollision (HANDLE m_HCom, UINT32 _uDriverParam, UINT32 _uStdParam, BYTE _bMaskBitLength, BYTE *_pMaskBytes, int *_pNumUIDs, BYTE *_pUidByteArray, int *_pNumCollisions, int *_pCollisionMaskBitLength, BYTE *_pCollidedSlotsByteArray);

Parameter	:	m_HCom – handle of port with RFID interface _uDriverParam – reserved, should be 0xFF _uStdParam – reserved, should be 0xFF _bMaskBitLength – length of Mask for anticollision procedure (in Bits) *_pMaskBytes – pointer to array of byte, which contains mask bits *_pNumUIDs – pointer to int, which contains the number of successfully read UIDs after successful completion of instruction *_pUidByteArray – pointer to array of byte, which contains the UID bytes after successful completion of instruction (8 Bytes / UID) *_pNumCollisions – pointer to int, which contains the number of slots that need further anticollision procedures (collision is detected) after successful completion of instruction. This represent also the number of “collided slot” present in _pCollidedSlotsByteArray) *_pCollisionMaskBitLength – pointer to int, which contains the number of bits representing slot information in _pCollidedSlotsByteArray after successful completion of instruction. Length is always multiple of 4. If length is not multiple of 8, the not initialized nibble is set to 0000 *_pCollidedSlotsByteArray – pointer to array of byte, which contains the slot collision masks after successful completion of instruction
Result	:	success – 0, collision detected – 0x25 error – see error code table
Description	:	Function to perform ISO15693 inventory anticollision procedure

This function is available up from driver version 0x10.0x59 for interfaces supporting transparent ISO15693 and ISO14443A,B functionality. Please contact microsensys for further support about transparent reader functionality.

HF – ISO15693 Infineon my-D™ specific TAG functions

This chapter describes functions available for communication with Infineon my-D™ transponders based on the standard ISO15693.

BYTE __stdcall APIENTRY __stdcall iso15693_myd_authenticate_a(HANDLE m_HCom,BYTE *pSelectedUID, BYTE param, BYTE *pRfu, BYTE cCode, BYTE CA, BYTE KA, BYTE *pR_Auth_CNT,BYTE *pR_VICC_RND)

Parameter	:	m_HCom – handle of port with RFID interface *pSelectedUID – has to be set to null / nil for common usage Param – rfu, should be set to zero *pRfu – rfu, should be set to null cCode – command code according SRF55VxxS-P data book CA – for information see SRF55VxxS-P data book KA – for information see SRF55VxxS-P data book *pR_Auth_CNT – pointer to array of byte, which contains interface data after successful completion of instruction (SRF55VxxS-P data book) *pR_VICC_RND – pointer to array of byte, which contains interface data after successful completion of instruction (SRF55VxxS-P data book)
Result	:	success - 0 error – see error table or SRF55VxxS-P data book
Description	:	my-D® specific command (see SRF55VxxS-P data book) **This function is available up from driver version 0x10.0x23.**

BYTE __stdcall APIENTRY __stdcall iso15693_myd_authenticate_b(HANDLE m_HCom,BYTE *pSelectedUID, BYTE param, BYTE *pRfu, BYTE NrOfStoredKey, BYTE *pAuth_CNT, BYTE *pVCD_RND, BYTE *pMAC, BYTE *pR_Auth_CNT,BYTE *pR_MAC)

Parameter	:	m_HCom – handle of port with RFID interface *pSelectedUID – has to be set to null / nil for common usage Param – should be set to zero NrOfStoredKey – rfu, should be set to zero *pRfu – rfu, should be set to null *pAuth_CNT – pointer to array of byte, for further information see SRF55VxxS-P data book *pVCD_RND – pointer to array of byte, for further information see SRF55VxxS-P data book *pMAC – pointer to array of byte, for further information see SRF55VxxS-P data book *pR_Auth_CNT – pointer to array of byte, which contains interface data after successful completion of instruction (SRF55VxxS-P data book) *pR_MAC – pointer to array of byte, which contains interface data after successful completion of instruction (SRF55VxxS-P data book)
Result	:	success - 0 error – see error table or SRF55VxxS-P data book
Description	:	my-D® specific command (see SRF55VxxS-P data book) **This function is available up from driver version 0x10.0x23.**

BYTE __stdcall APIENTRY __stdcall iso15693_myd_read_page(HANDLE m_HCom,BYTE *pSelectedUID, BYTE REQ_Byte, BYTE param, BYTE *pRfu, INT32 page_address, BYTE *pMAC, BYTE *pByteArray)

Parameter	:	m_HCom – handle of port with RFID interface *pSelectedUID – has to be set to null / nil for common usage REQ_Byte – request byte according ISO15693 Param – should be set to zero Adr0 – block address to read data from *pMAC – pointer to array of byte, for further information see SRF55VxxS-P data book (should be set to null, if not used) *pByteArray – pointer to array of byte, which contains interface data after successful completion of instruction (SRF55VxxS-P data book)
Result	:	success - 0 error – see error table or SRF55VxxS-P data book
Description	:	my-D® specific command (see SRF55VxxS-P data book) **This function is available up from driver version 0x10.0x23.**

BYTE __stdcall APIENTRY __stdcall iso15693_myd_write_page(HANDLE m_HCom,BYTE *pSelectedUID, BYTE REQ_Byte, BYTE param, BYTE *pRfu, INT32 page_address, BYTE *pMAC, BYTE *pByteArray)

Parameter	:	m_HCom – handle of port with RFID interface *pSelectedUID – has to be set to null / nil for common usage REQ_Byte – request byte according ISO15693 Param – should be set to zero Adr0 – block address to write data to *pMAC – pointer to array of byte, for further information see SRF55VxxS-P data book (should be set to null, if not used) *pByteArray – pointer to array of byte, which contains data, for further information see SRF55VxxS-P data book
Result	:	success - 0 error – see error table or SRF55VxxS-P data book
Description	:	my-D® specific command (see SRF55VxxS-P data book) **This function is available up from driver version 0x10.0x23.**

BYTE __stdcall APIENTRY __stdcall iso15693_myd_restricted_write_page(HANDLE m_HCom,BYTE *pSelectedUID, BYTE REQ_Byte, BYTE param, BYTE *pRfu, INT32 page_address, BYTE *pMAC, BYTE *pByteArray)

Parameter	:	m_HCom – handle of port with RFID interface *pSelectedUID – has to be set to null / nil for common usage REQ_Byte – request byte according ISO15693 Param – should be set to zero Adr0 – block address to write data to *pMAC – pointer to array of byte, for further information see SRF55VxxS-P data book (should be set to null, if not used) *pByteArray – pointer to array of byte, which contains data, for further information see SRF55VxxS-P data book
Result	:	success - 0 error – see error table or SRF55VxxS-P data book
Description	:	my-D® specific command (see SRF55VxxS-P data book) **This function is available up from driver version 0x10.0x23.**

BYTE __stdcall APIENTRY __stdcall iso15693_myd_write_and_reread_page(HANDLE m_HCom,BYTE *pSelectedUID, BYTE REQ_Byte, BYTE param, BYTE *pRfu, INT32 page_address, BYTE *pMAC, BYTE *pByteArray, BYTE *pResponse)

Parameter	:	m_HCom – handle of port with RFID interface *pSelectedUID – has to be set to null / nil for common usage REQ_Byte – request byte according ISO15693 Param – should be set to zero Adr0 – block address to write data to *pMAC – pointer to array of byte, for further information see SRF55VxxS-P data book (should be set to null, if not used) *pByteArray – pointer to array of byte, which contains data, for further information see SRF55VxxS-P data book *pResponse – pointer to array of byte, which contains interface data after successful completion of command, for further information see SRF55VxxS-P data book
Result	:	success - 0 error – see error table or SRF55VxxS-P data book
Description	:	my-D® specific command (see SRF55VxxS-P data book) **This function is available up from driver version 0x10.0x23.**

BYTE __stdcall APIENTRY __stdcall iso15693_myd_restricted_write_and_reread_page (HANDLE m_HCom,BYTE *pSelectedUID, BYTE REQ_Byte, BYTE param, BYTE *pRfu, INT32 page_address, BYTE *pMAC, BYTE *pByteArray, BYTE *pResponse)

Parameter	:	m_HCom – handle of port with RFID interface *pSelectedUID – has to be set to null / nil for common usage REQ_Byte – request byte according ISO15693 Param – should be set to zero Adr0 – block address to write data to *pMAC – pointer to array of byte, for further information see SRF55VxxS-P data book (should be set to null, if not used) *pByteArray – pointer to array of byte, which contains data, for further information see SRF55VxxS-P data book
Result	:	success - 0 error – see error table or SRF55VxxS-P data book
Description	:	my-D® specific command (see SRF55VxxS-P data book) **This function is available up from driver version 0x10.0x23.**

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 code 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 code table
Description : function writes single byte to my-D transponder near RFID interface (command is only available for my-D in custom mode)

HF – ISO15693-2 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 code 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 code 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 code 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 code 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 code table
Description	:	function sets lockbit for customer OTP area of iID®-G transponder (see transponder description) (Attention: operation is irreversible!)

This command will not be supported by future iID® contactless interfaces. Please use iidg_c_lock_block_16.

BYTE __stdcall APIENTRY __stdcall iidg_c_lock_block_16 (HANDLE m_HCom,BYTE BlockAddress_iIDG,BYTE Lock)

Parameter	:	m_HCom – handle of port with RFID interface BlockAddress – address of transponder to lock Lock : internal, should be set 1
Result	:	success - 0 error – see error code table
Description	:	function sets lock bit of iID®-G TAG block. Please note, that iID®-G -TAG allows lock of blocks 0 and 1 by hardware.

HF – ISO14443 iID®-L TAG specific functions

HF – ISO14443 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 successful completion of instruction, byte 0 contains length of data
Result	:	success - 0 error – see error code 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 code 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 code 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 successful completion of instruction, byte 0 contains length of data
Result	:	success - 0 error – see error code 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 code 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 code 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 code 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 code 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 code 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 successful completion of instruction, byte 0 contains length of data
Result	:	success - 0 error – see error code 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 code 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 successful completion of instruction, byte 0 contains length of data
Result	:	success - 0 error – see error code 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 code table
Description	:	function writes block (128Byte) to E²PROM of iID-L transponder near RFID interface (usage of command is depending on transponder configuration)

HF – ISO14443 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 code 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 code 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 iidl_c_get_sensor_information (HANDLE m_HCom, BYTE _Param, UINT32 _driverParam, UInt32 *_iSerNo, BYTE *_bPC, BYTE *_bPVer, BYTE *bNumOfSizes , BYTE *_bPSize, BYTE *pXtraData)

Parameter : m_HCom – handle of port with RFID interface
_Param – reserved, should be set 0
_driverParam – reserved, should be 0
_iSerNo – pointer to Unit32, which contains TELID serial number*
_bPC – pointer to byte, which contains TELID product code*
_bPVer – pointer to byte, which contains TELID product version*
_bNumOfSizes – pointer byte, which contains number of phys. sizes*
_bPSize – pointer to array of byte, which contains measured phys. sizes*
_bXtraData – pointer to array of byte, which contains additional sensor data (byte 0 contains data length, data is required to calculate sensor values using iidl_c_get_sensor_data)
*after successful completion of command
Result : success - 0
error – see error code table
Description : This function can be used to detect iID®-L based TELID® sensors including their basic parameter set

int __stdcall APIENTRY __stdcall iidl_c_get_sensor_data (HANDLE m_HCom, BYTE _Param, UINT32 _driverParam, UInt32 _iSerNo, BYTE _bPC, BYTE _bPVer, BYTE *bNumOfSizes , BYTE *_bPSize , BYTE *pXtraData, BYTE _bNumOfSensorValues, float *_fSensorValues)

Parameter	:	m_HCom – handle of port with RFID interface _Param – reserved, should be set 0 _driverParam – reserved, should be 0 _iSerNo – TELID serial number* _bPC –TELID product code* _bPVer –TELID product version* _bNumOfSizes – pointer byte, which contains number of phys. sizes* _bPSize – pointer to array of byte, which contains measured phys. sizes* _bNumOfSensorValues – pointer byte, which contains number of sensor values _fSensorValues – pointer to array of float, which contains measurement values _bXtraData – array of byte, which contains additional sensor data (byte 0 contains data length) *see iidl_c_get_sensor_information
Result	:	success - 0 error – see error code table
Description	:	This function can be used to get sensor values of iID®-L based TELID® sensors. Please note, that one additional call of iidl_c_get_sensor_information is required before.

int __stdcall APIENTRY __stdcall iidl_c_get_sensor_data_extended (HANDLE m_HCom, UINT32 _driverParam, BYTE _bPhysicalSize, BYTE _bSensorType, BYTE _bParamLength, BYTE *_bParams, UINT32 _iSerNo, BYTE _bPC, BYTE _bPVer, BYTE *_bNumOfSizes, BYTE *_bPSize, BYTE *pXtraData, BYTE *_bNumOfSensorValues, float *_fSensorValues);

Parameter	:	see iidl_c_get_sensor_data _bPhysicalSize – physical size to read _bSensorType – sensor type of TELID®2xx _pParamLength – number of parameter bytes _bParams – pointer to array of bytes containing parameters _iSerNo – TELID serial number*
Result	:	success - 0 error – see error code table
Description	:	This function can be used to get special sensor values of iID®-L based TELID® sensors by definition of sensortype, physical size and extra params. Please note, that one additional call of iidl_c_get_sensor_information is required before. Please contact microsensys for further information or sample codes.

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 code 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 code 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 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 successful
Description : Command is reading the temperature from any iID-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 successful
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 successful
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 successful
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 successful
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.

HF – ISO14443A - Mifare UltraLight ™ TAG functions

This chapter describes functions available for communication with transponders compatible to Mifare UltraLight command set.

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 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 code 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 successful completion of instruction, byte 0 contains length of data
Result	:	success - 0 error – see error code 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 code table
Description	:	function writes block to Mifare UltraLight ™ transponder near RFID interface

HF – ISO14443A standard and NXP N-TAG functions

This chapter describes functions available for communication with transponders compatible to ISO/IEC 14443 A communication standard.
Functions are available up from driver version 0x10.0x4E and require ISO14443A transparent command reader implementation.

BYTE __stdcall APIENTRY __stdcall iso14443a_read_transparent(HANDLE m_HCom, UINT32 _uDriverParam, UINT32 _uStdParam, BYTE _bBlockNum, BYTE *_bData)

Parameter	:	m_HCom – handle of port with RFID interface _uDriverParam – reserved, should be 0 _uStdParamParam – reserved, should be 0 _bBlockNum – block number to read data *_bData – 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 code table
Description	:	function reads data from ISO1443A compatible transponder near RFID interface

BYTE __stdcall APIENTRY __stdcall iso14443a_write_transparent(HANDLE m_HCom, UINT32 _uDriverParam, UINT32 _uStdParam, BYTE _bBlockNum, BYTE *_bData)

Parameter	:	m_HCom – handle of port with RFID interface _uDriverParam – reserved, should be 0 _uStdParamParam – reserved, should be 0 _bBlockNum – block number to write data *_bData – pointer to array of byte, which contains data to write
Result	:	success - 0 error – see error code table
Description	:	function writes data to ISO1443A compatible transponder near RFID interface

BYTE __stdcall APIENTRY __stdcall ntag_password_auth(HANDLE m_HCom, UINT32 _uDriverParam, UINT32 _uStdParam, BYTE *_bPassword, int *_iPACK)

Parameter	:	m_HCom – handle of port with RFID interface _uDriverParam – reserved, should be 0 _uStdParamParam – reserved, should be 0 *_bPassword – pointer to array of byte, which contains password (4Bytes) *_iPack – pointer to integer, which contains NXP N-TAG pack value (2 bytes) after successful completion of command
Result	:	success - 0 error – see error code table
Description	:	function performs password verification using PWD_AUTH (command code 0x1B function of NXP N-TAG transponder near RFID interface

HF – ISO14443 iID®-K TAG functions

BYTE __stdcall APIENTRY __stdcall iidk_c_read_uid (HANDLE m_HCom,INT32 DriverParam,INT32 ReaderParam, BYTE *pByteArray)

Parameter	:	m_HCom – handle of port with RFID interface INT32 DriverParam – reserved, should be 0xFF INT32 ReaderParam – reserved, should be 0x00 *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 code table
Description	:	function reads UID-Code from iID-K transponder near RFID interface

BYTE __stdcall APIENTRY __stdcall iidk_c_write_linear (HANDLE m_HCom,INT32 BlockAddress,INT32 DriverParam, INT32 ReaderParam,BYTE *pByteArray)

Parameter	:	m_HCom – handle of port with RFID interface BlockAddress – address of transponder to write data to INT32 DriverParam – reserved, should be 0xFF INT32 ReaderParam – reserved, should be 0x00 *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 code table
Description	:	function writes block (32Byte) to iID-K transponder near RFID interface

BYTE __stdcall APIENTRY __stdcall iidk_c_read_linear (HANDLE m_HCom,INT32 BlockAddress,INT32 DriverParam, INT32 ReaderParam,BYTE *pByteArray)

Parameter	:	m_HCom – handle of port with RFID interface BlockAddress – address of transponder to read data from INT32 DriverParam – reserved, should be 0xFF INT32 ReaderParam – reserved, should be 0x00 *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 code table
Description	:	function reads block (32Byte) from iID-K transponder near RFID interface

HF - LEGIC® Prime TAG functions

This chapter describes functions available for communication with transponders based on the LEGIC RFID transponders, type 'Prime'. Additional LEGIC Prime/ Advant specific function definitions are available on request.

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 code table
Description	:	function reads serial number from LEGIC® Prime transponder near RFID interface

BYTE __stdcall APIENTRY __stdcall legic_c_read_segment (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 param = 0 -> read segment 1, other values are not defined *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 code table
Description	:	function reads segment from LEGIC® Prime transponder near RFID interface

BYTE __stdcall APIENTRY __stdcall legic_c_read_data (HANDLE m_HCom, BYTE param, BYTE* pType, BYTE* pByteArray); [2]

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 code table
Description	:	function reads custom specific data from LEGIC® Prime transponder near RFID interface

[1] customized command – please ask MICROSENSYS for further information
[2] customized command – please ask MICROSENSYS for further information

HF - Inside Contactless™ Pico-TAG functions

This chapter describes functions available for communication with transponders of Inside Contactless Pico-TAG system.

BYTE __stdcall APIENTRY __stdcall picotag_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 code table
Description	:	function reads UID from Pico-TAG transponder near RFID interface

BYTE __stdcall APIENTRY __stdcall picotag_c_read_block_8 (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 : internal, should be set 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 code table
Description	:	function reads block from Pico-TAG transponder near RFID interface

BYTE __stdcall APIENTRY __stdcall picotag_c_write_block_8 (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 : internal, should be set 0 *pByteArray – pointer to array of byte, which contains data to write to transponder
Result	:	success - 0 error – see error code table
Description	:	function writes block to Pico-TAG transponder near RFID interface

BYTE __stdcall APIENTRY __stdcall picotag_c_lock_block_8 (HANDLE m_HCom,INT32 BlockAddress,BYTE Lock)

Parameter	:	m_HCom – handle of port with RFID interface BlockAddress – address of transponder to lock Lock : internal, should be set 1
Result	:	success - 0 error – see error code table
Description	:	function sets lock bit of pico-TAG block. Please note, that pico-TAG allows lock of blocks 6..12 by hardware.

HF - echo functions

This chapter describes special functions available for HF communication with dual interface transponders EM4423, so called EM echo.

BYTE __stdcall APIENTRY __stdcall echo_read_uhf_epc (HANDLE m_HCom, UINT32 _uDriverParam, BYTE bParamByte, BYTE *pByteArray);

Parameter	:	m_HCom – handle of port with RFID interface _uDriverParam – reserved, should be 0xFF bParamByte – output format 0 → Automatic. If ISO-bit is set, UID in ASCII, otherwise UII in HEX 1 → PC+EPC in HEX 2 → UID in ASCII *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 code table
Description	:	function reads transponder EPC code (same for UHF frontend) of an EM echo dual frequency transponder near RFID interface

This function is available up from driver version 0x10.0x59 for interfaces supporting ECHO read EPC functionality. Please contact microsensys for further support about transparent reader functionality.

BYTE __stdcall APIENTRY __stdcall echo_read_uhf_uid (HANDLE m_HCom, UINT32 _uDriverParam, BYTE *pByteArrayRaw, BYTE *pByteArrayDecoded);

Parameter	:	m_HCom – handle of port with RFID interface _uDriverParam – reserved, should be 0xFF *pByteArrayRaw – pointer to array of byte, which contains PC+EPC data after successful completion of instruction, byte 0 contins length of data *pByteArrayDecoded – pointer to array of byte, which contains UID in ASCII format, if the read PC+EPC represents an ISO UID after successful completion of instruction, byte 0 contains length of data
Result	:	success - 0 error – see error code table
Description	:	function reads transponder EPC code (same for UHF frontend) of an EM echo dual frequency transponder near RFID interface, checks the ISO UID flag, and in case it is set converts the bytes to its ASCII representation

This function is available up from driver version 0x10.0x59 for interfaces supporting ECHO read EPC functionality. Please contact microsensys for further support about transparent reader functionality.

HF - iID®-A sensor functions (obsolete)

BYTE __stdcall APIENTRY __stdcall c_iida_get_temperature (HANDLE m_HCom, BYTE *pSelectedUID, BYTE REQ_Byte, BYTE param, BYTE *pResponseData, double *temperature)

Parameter	:	*m_Hcom* - handle of port to be worked with *pSelectedUID - pointer to byte array containing UID when working in selected mode REQ_Byte – request byte according ISO15693 Param – should be 0x03 for normal usage *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 iID-L transponder near receiver.

HF - I-Code® UID TAG functions (obsolete)

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 successful completion of instruction, byte 0 contains length of data (5)
Result : success - 0
error – see error code 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 specifcation SL2 ICS11 Rev 3.0/2004 Jan 30
- others – reserved for future development
*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 code 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 specifcation 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 code 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 code table
Description	:	function writes 1 byte to I-Code® UID transponder near RFID interface without CRC generation

HF - I-Code® 1 TAG functions (obsolete)

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 successful completion of instruction, byte 0 contains length of data
Result	:	success - 0 error – see error code 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 successful completion of instruction, byte 0 contains length of data
Result	:	success - 0 error – see error code 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 code 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 code 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 code 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 code 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 code table
Description	:	function locks block of I-Code®1 transponder near RFID interface (Att.: This command is irreversible !)

HF - iID®-D TAG functions (obsolete)

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);

UHF transponder chip specific driver functions

UHF - ISO18000-6c TAG functions

BYTE __stdcall APIENTRY __stdcall iso18000_6_c_read_epc_gen2 (HANDLE m_HCom, UINT32 _uDriverParam, BYTE *_bNumIDs, BYTE *_bTagTypes, BYTE *_bLengthIDs, BYTE *_bData);

Parameter	:	m_HCom – handle of port with RFID interface UINT32 _uDriverParam – reserved, should be 0xFF BYTE *_bNumIDs – pointer to byte, which contains number of codes read BYTE *_bTagTypes – pointer to array of byte, which contains TAG types of TAGs read BYTE *_bLengthIDs – pointer to array of byte, which contains code length of TAGs read BYTE *_bData – pointer to array of byte, which contains code data after successful completion of instruction
Result	:	success - 0 error – see error code table
Description	:	function reads EPC from transponders near RFID interface. Maximum length per EPC supported by this library is 64bytes. See *c_set_EPC_extended_parameter* for additional options.

BYTE __stdcall APIENTRY __stdcall iso18000_6_c_read_epc_rssi_gen2 (HANDLE m_HCom, UINT32 _uDriverParam, BYTE *_bNumIDs, BYTE *_bTagTypes, BYTE *_bLengthIDs, BYTE *_bData, float *rssiValues);

Parameter	:	m_HCom – handle of port with RFID interface UINT32 _uDriverParam – reserved, should be 0xFF BYTE *_bNumIDs – pointer to byte, which contains number of codes read BYTE *_bTagTypes – pointer to array of byte, which contains TAG types of TAGs read BYTE *_bLengthIDs – pointer to array of byte, which contains code lengths of different TAGs read BYTE *_bData – pointer to array of byte, which contains code data after successful completion of instruction float *rssiValues – pointer to array of float, which contains RSSI values related to EPC codes located in *_bData after successful completion of instruction
Result	:	success - 0 error – see error code table
Description	:	function reads EPC and related RSSI values from transponders near RFID interface. Maximum length per EPC supported by this library is 64bytes. See *c_set_EPC_extended_parameter* for additional options. This function requires reader firmware version 0x98.05.00 or higher.

BYTE __stdcall APIENTRY __stdcall iso18000_6_c_write_epc_gen2 (HANDLE m_HCom, UINT32 _uDriverParam, UINT32 _uStdParam, BYTE *_bCode, BYTE _bCodeLength, BYTE *_bNewPcBytes, BYTE *_bNewEpcBytes, BYTE _bNewEpcLength, BYTE *_bPassword);

Parameter : m_HCom – handle of port with RFID interface
UINT32 _uDriverParam – reserved, should be 0xFF
UINT32 _uStdParam – hardware parameter, see hardware documentation
BYTE *_bCode – pointer to array of byte containing current EPC code
BYTE _bCodeLength –byte, which contains length of current EPC
BYTE *_bNewPCBytes – pointer to array of byte containing new PC bytes
BYTE *_bNewEPCBytes – pointer to array of byte containing new EPC code
BYTE _bNewEPCLength –byte, which contains length of new EPC code
BYTE *_bPassword – pointer to array of byte containing password (H...L), null for programming without password
Result : success - 0
error – see error code table
Description : function writes data to transponder near RFID interface

BYTE __stdcall APIENTRY __stdcall c_set_EPC_extended_parameter(BYTE _param)

Parameter : _param – new EPC scan setting

Data content (binary)	Description
xxxx xxx1	EPC scan response keeps PC word during EPC scan
xxxx xxx0 (default)	No return of PC word during EPC scan
xxxx xx1x	EPC scan response keeps session during EPC scan
xxxx xx0x (default)	No return of session during EPC scan
xxxx x1xx	EPC scan response keeps RSSI value during EPC scan (reader firmware version up to 0x98.04.xx, see iso18000_6_c_read_epc_rssi_gen2 for new implementation)
xxxx x0xx (default)	No return of RSSI value during EPC scan

Result : current driver setting (see param)
Description : function defines new EPC scan setting. *This function is available up from driver version 0x10.0x4F.*

BYTE __stdcall APIENTRY __stdcall c_get_EPC_extended_parameter()

Parameter : -
Result : current driver setting (see c_set_EPC_extended_parameter)
Description : function returns current EPC scan setting. *This function is available up from driver version 0x10.0x4F.*

BYTE __stdcall APIENTRY __stdcall c_set_AFI_mask (BYTE _afi)

Parameter : _afi – new AFI bit mask according ISO18000-6c. Default AFI mask is 0.
Result : current driver setting (see _afi)
Description : function defines new EPC scan setting. *This function is available up from driver version 0x10.0x4F.*

BYTE __stdcall APIENTRY __stdcall c_get_AFI_mask(BYTE *_afi)

Parameter	:	BYTE *_afi – pointer to byte containing current driver AFI mask setting (see c_set_AFI_mask). Default AFI mask is 0.
Result	:	success - 0 error – see error code table
Description	:	function returns current driver AFI setting. *This function is available up from driver version 0x10.0x4F.*

BYTE __stdcall APIENTRY __stdcall iso18000_6_c_read_words (HANDLE m_HCom, UINT32 _uDriverParam, UINT32 _uStdParam, UINT32 _uDataBank, UINT32 _uDataAddress, UINT32 _uDataLength, BYTE *_bCode, BYTE _bCodeLength, BYTE *_bData);

Parameter	:	m_HCom – handle of port with RFID interface UINT32 _uDriverParam – reserved, should be 0xFF UINT32 _uStdParam – hardware parameter, see hardware documentation UINT32 _uDataBank – page address to read from transponder UINT32 _uDataAddress – memory address to read from transponder UINT32 _uDataLength – data length to read from transponder BYTE *_bCode – pointer to byte containing EPC code BYTE _bCodeLength –byte, which contains length of EPC BYTE *_bData – pointer to array of byte, which contains data after successful completion of instruction, byte 0 contains length of data
Result	:	success - 0 error – see error code table
Description	:	function reads data from transponder near RFID interface

BYTE __stdcall APIENTRY __stdcall iso18000_6_c_read_words_secure (HANDLE m_HCom, UINT32 _uDriverParam, UINT32 _uStdParam, UINT32 _uDataBank, UINT32 _uDataAddress, UINT32 _uDataLength, BYTE *_bCode, BYTE _bCodeLength, BYTE *_bPassword, BYTE *_bData);

Parameter	:	m_HCom – handle of port with RFID interface UINT32 _uDriverParam – reserved, should be 0xFF UINT32 _uStdParam – hardware parameter, see hardware documentation UINT32 _uDataBank – page address to read from transponder UINT32 _uDataAddress – memory address to read from transponder UINT32 _uDataLength – data length to read from transponder BYTE *_bCode – pointer to byte containing EPC code BYTE _bCodeLength –byte, which contains length of EPC BYTE *_bPassword – pointer to array of byte containing password (H...L) BYTE *_bData – pointer to array of byte, which contains data after successful completion of instruction, byte 0 contains length of data
Result	:	success - 0 error – see error code table
Description	:	function performs secured transponder READ operation including secured ACCESS function

BYTE __stdcall APIENTRY __stdcall iso18000_6_c_write_words (HANDLE m_HCom, UINT32 _uDriverParam, UINT32 _uStdParam, UINT32 _uDataBank, UINT32 _uDataAddress, UINT32 _uDataLength, BYTE *_bCode, BYTE _bCodeLength, BYTE *_bData);

Parameter	:	m_HCom – handle of port with RFID interface UINT32 _uDriverParam – reserved, should be 0xFF UINT32 _uStdParam – hardware parameter, see hardware documentation UINT32 _uDataBank – page address to read from transponder UINT32 _uDataAddress – memory address to read from transponder UINT32 _uDataLength – data length to read from transponder BYTE *_bCode – pointer to byte containing EPC code BYTE _bCodeLength –byte, which contains length of EPC BYTE *_bData – pointer to array of byte, which contains data to write
Result	:	success - 0 error – see error code table
Description	:	function writes data to transponder near RFID interface

BYTE __stdcall APIENTRY __stdcall iso18000_6_c_write_words_secure (HANDLE m_HCom, UINT32 _uDriverParam, UINT32 _uStdParam, UINT32 _uDataBank, UINT32 _uDataAddress, UINT32 _uDataLength, BYTE *_bCode, BYTE _bCodeLength, BYTE *_bPassword, BYTE *_bData);

Parameter	:	m_HCom – handle of port with RFID interface UINT32 _uDriverParam – reserved, should be 0xFF UINT32 _uStdParam – hardware parameter, see hardware documentation UINT32 _uDataBank – page address to read from transponder UINT32 _uDataAddress – memory address to read from transponder UINT32 _uDataLength – data length to read from transponder BYTE *_bCode – pointer to byte containing EPC code BYTE _bCodeLength –byte, which contains length of EPC BYTE *_bPassword – pointer to array of byte containing password (H...L) BYTE *_bData – pointer to array of byte, which contains data to write
Result	:	success - 0 error – see error code table
Description	:	function performs secured transponder WRITE operation including secured ACCESS function

BYTE __stdcall APIENTRY __stdcall iso18000_6_c_lock(HANDLE m_HCom, UINT32 _uDriverParam, UINT32 _uStdParam, UINT32 _uPayload, BYTE *_bCode, BYTE _bCodeLength)

Parameter	:	m_HCom – handle of port with RFID interface UINT32 _uDriverParam – reserved, should be 0xFF UINT32 _uStdParam – hardware parameter, see hardware documentation UINT32 _uPayLoad – transponder specific parameter, see ISO18000-6c / EPCGlobal Gen2 protocol definition for details BYTE *_bCode – pointer to array of byte containing EPC code BYTE _bCodeLength –byte, which contains length of EPC
Result	:	success - 0 error – see error code table
Description	:	function performs lock command procedure on RFID-tag near RFID interface.

BYTE __stdcall APIENTRY __stdcall iso18000_6_c_lock_secure (HANDLE m_HCom, UINT32 _uDriverParam, UINT32 _uStdParam, UINT32 _uPayLoad, BYTE *_bCode, BYTE _bCodeLength, BYTE *_bPassword);

Parameter	:	m_HCom – handle of port with RFID interface UINT32 _uDriverParam – reserved, should be 0xFF UINT32 _uStdParam – hardware parameter, see hardware documentation UINT32 _uPayLoad – transponder specific parameter, see ISO18000-6c / EPCGlobal Gen2 protocol definition for details BYTE *_bCode – pointer to byte containing EPC code BYTE _bCodeLength –byte, which contains length of EPC BYTE *_bPassword – pointer to array of byte containing password (H...L)
Result	:	success - 0 error – see error code table
Description	:	function performs secured transponder LOCK operation including secured ACCESS function

BYTE __stdcall APIENTRY __stdcall iso18000_6_c_kill(HANDLE m_HCom, UINT32 _uDriverParam, UINT32 _uStdParam, BYTE *_bPassword, BYTE _paramRightAdjusted, BYTE *_bCode, BYTE _bCodeLength)

Parameter	:	m_HCom – handle of port with RFID interface UINT32 _uDriverParam – reserved, should be 0xFF UINT32 _uStdParam – hardware parameter, see hardware documentation BYTE *_bPassword – pointer to array of byte containing KILL password (H..L) UINT32 _paramRightAdjusted – transponder specific parameter for KILL procedure, see ISO18000-6c for details BYTE *_bCode – pointer to array of byte containing EPC code BYTE _bCodeLength –byte, which contains length of EPC
Result	:	success - 0 error – see error code table
Description	:	function performs KILL command procedure on RFID-tag near RFID interface.

UHF - ISO18000-6c transparent functions and sensor TAG functions

BYTE __stdcall APIENTRY __stdcall c_transparent_UHF_open(HANDLE m_HCom, UINT32 _uStdParam, UINT32 _uDriverParam, BYTE _bCodeLength, BYTE *_bCode, BYTE *_bCustomDataRcvd)

Parameter	:	m_HCom – handle of port with RFID interface UINT32 _uStdParam – hardware parameter, see hardware documentation UINT32 _uDriverParam – reserved, should be 0xFF BYTE *_bCode – pointer to byte containing EPC code BYTE _bCodeLength –byte, which contains length of EPC BYTE *_bCustomData – custom data to send to transponder BYTE *_bCustomDataRcvd – custom data received from transponder after successful completion of instruction, byte 0 contains length of data
Result	:	success - 0 error – see error code table
Description	:	function performs transparent ISO18000-6c OPEN for transponder near RFID interface.

BYTE __stdcall APIENTRY __stdcall c_transparent_UHF_execute(HANDLE m_HCom, UINT32 _uStdParam, UINT32 _uDriverParam, BYTE _bTxType, int _iExpectedAnswerLength, BYTE _bBrokenBits, BYTE _bBytesToSend, BYTE *_bCustomData, bool _appendRN16, BYTE *_bRN16, BYTE *_bCustomDataRcvd)

Parameter	:	m_HCom – handle of port with RFID interface UINT32 _uStdParam – hardware parameter, see hardware documentation UINT32 _uDriverParam – reserved, should be 0xFF BYTE _bTXType –byte containing TX type int _iExpectedAnswerLength – byte length of expected answer BYTE _bBytesToSend – number of custom bytes to send to transponder BYTE _bBrokenBits – number of custom broken bits to send to transponder BYTE *_bCustomData – pointer to array of byte containing custom data to send to transponder bool _appendRN16 – defines, whether RN16 should be append automatically *_bRN16 – pointer to array of byte containing RN16 to send BYTE *_bCustomDataRcvd – custom data received from transponder after successful completion of instruction, byte 0 contains length of data
Result	:	success - 0 error – see error code table
Description	:	function performs transparent ISO18000-6c operation for transponder near RFID interface.

BYTE __stdcall APIENTRY __stdcall c_transparent_UHF_close(HANDLE m_HCom, UINT32 _uStdParam, UINT32 _uDriverParam)

Parameter	:	m_HCom – handle of port with RFID interface UINT32 _uStdParam – hardware parameter, see hardware documentation UINT32 _uDriverParam – reserved, should be 0xFF
Result	:	success - 0 error – see error code table
Description	:	function performs transparent ISO18000-6c CLOSE for transponder near RFID interface.

BYTE __stdcall APIENTRY __stdcall epc_c_custom(HANDLE m_HCom, UINT32 _uDriverParam, UINT32 _uStdParam, BYTE *_bCode, BYTE _bCodeLength, BYTE *_bCustomData, BYTE *_bCustomDataRcvd);

Parameter	:	m_HCom – handle of port with RFID interface UINT32 _uDriverParam – reserved, should be 0xFF UINT32 _uStdParam – hardware parameter, see hardware documentation BYTE *bCode – pointer to byte containing EPC code BYTE _bCodeLength –byte, which contains length of EPC BYTE *_bCustomData – custom data to send to transponder BYTE *_bCustomDataReceived – custom data received from transponder after successful completion of instruction, byte 0 contains length of data
Result	:	success - 0 error – see error code table
Description	:	function performs custom commands on wireless interface to transponder near RFID interface. epc_c_custom automatically performs OPEN, EXECUTE and CLOSE custom commands using epc_c_direct_custom_command.

BYTE __stdcall APIENTRY __stdcall epc_c_direct_custom_command(HANDLE m_HCom, UINT32 _uDriverParam, UINT32 _uStdParam, BYTE *_bCode, BYTE _bCodeLength, BYTE _bCustomType, BYTE *_bCustomData, BYTE *_bCustomDataRcvd);

Parameter	:	m_HCom – handle of port with RFID interface UINT32 _uDriverParam – reserved, should be 0xFF UINT32 _uStdParam – hardware parameter, see hardware documentation BYTE *bCode – pointer to byte containing EPC code BYTE _bCodeLength –byte, which contains length of EPC BYTE _bCustomType – type of custom command BYTE *_bCustomData – custom data to send to transponder BYTE *_bCustomDataReceived – custom data received from transponder after successful completion of instruction, byte 0 contains length of data
Result	:	success - 0 error – see error code table
Description	:	function performs custom commands on wireless interface to transponder near RFID interface

BYTE __stdcall APIENTRY __stdcall c_get_temperature_intprotv4(HANDLE m_HCom, UINT32 _uDriverParam, UINT32 _uStdParam, UINT32 _uTagType, UINT32 _uSensorType, BYTE *_bCode, BYTE _bCodeLength, double *_dTemperatureValue, BYTE *_bData);

Parameter	:	m_HCom – handle of port with RFID interface UINT32 _uDriverParam – reserved, should be 0xFF UINT32 _uStdParam – hardware parameter, see hardware documentation UINT32 _uTagType – page address to read from transponder UINT32 _uSensorType – memory address to read from transponder BYTE *_bCode – pointer to byte containing EPC code BYTE _bCodeLength –byte, which contains length of EPC double *dTemperatureValue – pointer to double containing temperature data after successful completion of instruction BYTE *bData – pointer to array of byte, which contains raw data after successful completion of instruction, byte 0 contains length of data
Result	:	success - 0 error – see error code table
Description	:	function reads temperature data to transponder near RFID interface

LF transponder chip specific driver functions

LF - 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 successful completion of instruction, byte 0 contains length of data
Result	:	success - 0 error – see error code 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 code 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 successful completion of instruction, byte 0 contains length of data
Result	:	success - 0 error – see error code 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 successful completion of instruction, byte 0 contains length of data
Result	:	success - 0 error – see error code 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 successful completion of instruction, byte 0 contains length of data
Result	:	success - 0 error – see error code 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 successful completion of instruction, byte 0 contains length of data
Result	:	success - 0 error – see error code table
Description	:	function reads Identifier from special PSK transponder near RFID interface (please ask MICROSENSYS for more information)

Custom reader driver functions

LDR02 Reader functions

Following described are some functions available for communication with LDR02 reader.

BYTE __stdcall APIENTRY __stdcall c_ldr02_scan_start(HANDLE m_HCom, BYTE _bType, BYTE _bBreakTime);

Parameter	:	m_HCom – handle of port with RFID interface _bType – scan type according to LDR02 wire interface _bBreakTime – break time according to LDR02 wire interface
Result	:	success - 0 error – see error code table
Description	:	function starts the LDR02 internal scan functionality

BYTE __stdcall APIENTRY __stdcall c_ldr02_scan_stop(HANDLE m_HCom);

Parameter	:	m_HCom – handle of port with RFID interface
Result	:	success - 0 error – see error code table
Description	:	function stops the LDR02 internal scan functionality

BYTE __stdcall APIENTRY __stdcall c_ldr02_scan_get_result(HANDLE m_HCom, bool _preClear, bool _postClear, BYTE *_bResultData);

Parameter	:	m_HCom – handle of port with RFID interface _preClear – deletes the available data from input buffer before trying to get result data _postClear – deletes the incoming data from input buffer after obtaining result data *pResultData – 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 code table
Description	:	function reads transmitted UID from LDR02 RFID reader

Pocket Reader iID®21x0 functions

iID® POCKETmini functions

Following described are some functions available for communication with the iID®21x0 MPC device types.

int __stdcall APIENTRY __stdcall iid2100_c_get_datacount (HANDLE m_HCom,BYTE ApplicationParam,int *DataSetCount,int *DataSetNumber);

Parameter	:	*m_Hcom* - handle of port to be worked with ApplicationParam - parameter byte for application control DataSetCount - address of integer value to store number of collected datasets DataSetNumber - address of integer value to store number of transferred datasets
Result	:	returns 0 if successful
Description	:	Command is reading the number of available datasets from iID®21x0. Please see hardware documentation for further information.

int __stdcall APIENTRY __stdcall iid2100_c_get_dataset (HANDLE m_HCom,BYTE ApplicationParam,int DataSetNumber, int Count, BYTE* pByteArray);

Parameter	:	*m_Hcom* - handle of port to be worked with ApplicationParam - parameter byte for application control DataSetNumber - integer value with datasetnumber for transfer start Count - number of datasets to transfer by function call (maximum count is depending on device type) *pByteArray – pointer to array of byte, which contains dataset(s) after successful completion of instruction, byte 0 contains length of data
Result	:	returns 0 if successful
Description	:	Command is reading dataset(s) from iID®21x0. Please see hardware documentation for further information about dataset structure.

int __stdcall APIENTRY __stdcall iid2100_c_reset_datasetcount (HANDLE m_HCom,BYTE ApplicationParam);

Parameter	:	*m_Hcom* - handle of port to be worked with ApplicationParam - parameter byte for application control
Result	:	returns 0 if successful
Description	:	Command resets the the number of available datasets to zero. Please see hardware documentation for further information.

int __stdcall APIENTRY __stdcall iid2100_c_set_datasetnumber (HANDLE m_HCom,BYTE ApplicationParam,int DataSetNumber);

Parameter	:	*m_Hcom* - handle of port to be worked with ApplicationParam - parameter byte for application control DataSetNumber - integer value with datasetnumber for next transfer
Result	:	returns 0 if successful
Description	:	Command sets the the number of read datasets to the actual value. Please see hardware documentation for further information.

int __stdcall APIENTRY __stdcall iid2100_c_set_parameters (HANDLE m_HCom,S_iID2100_Settings_USER S_Settings);

Parameter : *m_Hcom* - handle of port to be worked with
S_Settings – struct containing iID2110 parameter dataset

```
struct S_iID2100_Settings_USER
{
        INT32 MemorySize_User;
        byte ScanTime_sec;
        byte LEDTime_100msec;
        byte AutoOffTime_min;
        byte NrOfReadingsPerScan;
        byte KeyID_Red;
        byte KeyID_Blue;
        INT32 SystemMask;
        byte Reserved1;
        byte Reserved2;
        byte Reserved3;
        byte Counter_SaveDataSet;
        byte FirmwareMode;
} iID2100_Settings_USER;
```

Result : returns 0 if successful
Description : Command sets iID2110 hardware parameters. Please see hardware documentation for further information.

int __stdcall APIENTRY __stdcall iid2100_c_get_parameters (HANDLE m_HCom,S_iID2100_Settings_FULL *pSF,BYTE *pByteArray);

Parameter : *m_Hcom* - handle of port to be worked with
pSF – pointer to struct containing iID2110 parameter dataset

```
struct S_iID2100_Settings_FULL
{
        INT32 MemorySize;
        INT32 MemorySize_User;
        byte ScanTime_sec;
        byte LEDTime_100msec;
        byte AutoOffTime_min;
        byte NrOfReadingsPerScan;
        byte KeyID_Red;
        byte KeyID_Blue;
        INT32 SystemMask;
        byte Reserved1;
        byte Reserved2;
        byte Reserved3;
        byte Counter_SaveDataSet;
        byte RolloverMode;
        byte FirmwareMode;
        byte FirmwareType;
        byte FirmwareRev;
} iID2100_Settings_FULL;
```

pByteArray - pointer to array of byte, which contains interface data after successful completion of instruction, byte 0 contains length of data Result
: returns 0 if successful
Description : Command reads iID2110 hardware parameters. Please see hardware documentation for further information.

int __stdcall APIENTRY __stdcall iid2100_c_set_time (HANDLE m_HCom,int year, int month, int day, int hour, int minute, int second, int msec)

Parameter	:	*m_Hcom* - handle of port to be worked with Year, month, day, hour, minute, second, msec - parameters containing new Date/time to be transferred to iID®2110
Result	:	returns 0 if successful
Description	:	Command sets date/time to internal RTC of Pocket mini iID®2110. Please see hardware documentation for further information.

int __stdcall APIENTRY __stdcall iid2100_c_get_time (HANDLE m_HCom,int *pyear, int *pmonth, int *pday, int *phour, int *pminute, int *psecond, int *pmsec)

Parameter	:	*m_Hcom* - handle of port to be worked with *Year, *month, *day, *hour, *minute, *second, *msec – pointer to integer containing device date/time after successful completion of command
Result	:	returns 0 if successful
Description	:	Command gets date/time from internal RTC of Pocket mini iID®2110. Please see hardware documentation for further information.

iID® POCKETwork MPC functions

Following described are some functions available for communication with the iID® POCKETwork device types.

BYTE __stdcall APIENTRY __stdcall c_read_DataMemory_intprotv4(HANDLE m_HCom, BYTE _StdParam, UINT32 _driverParam, UINT32 _uDataAddress, BYTE _bLength, BYTE *_bData)

Description : internal function, not for general usage. See iiD® MPC library for further details of MPC communication.

BYTE __stdcall APIENTRY __stdcall c_update_DataMemory_intprotv4(HANDLE m_HCom, BYTE _StdParam, UINT32 _driverParam, UINT32 _uDataAddress, BYTE _bLength, BYTE *_bData)

Description : internal function, not for general usage. See iiD® MPC library for further details of MPC communication.

BYTE __stdcall APIENTRY __stdcall c_erase_DataMemory_intprotv4(HANDLE m_HCom, BYTE _StdParam, UINT32 _driverParam, UINT32 _uDataAddress, BYTE _bEraseParam)

Description : internal function, not for general usage. See iiD® MPC library for further details of MPC communication.

Appendix

Error - Codetable

State	Error
0x00	no error
RFID interface error codes	
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
others	Other hardware specific error codes may occur – see hardware documentation.
Driver error codes	
0x08	identifiers do not match
0x11	range error
0x14	General TAG-error
0x35	No more identifiers
0x3F	Communication or port error
0x43	Configuration error
0x4F	unknown/not supported option detected
0xFF	general communication or driver error

See hardware documentation for further error codes.

Transponder systems

Following tables contain lists of transponder systems, supported by c_get_transponder_parameters type detection based on driver version 0x10.0x4E.

Code	*Transponder system*
0x01	125KHZ_UNIQUE
0x02	125KHZ_TITAN
0x04	125KHZ_PSK
0x16	125KHZ_FDXB
0x01	1356MHZ_ISO15693
0x02	1356MHZ_IID-L
0x04	1356MHZ_IID-D
0x08	1356MHZ_IID-G
0x10	1356MHZ_IID-N
0x20	1356MHZ_I-CODE-UID
0x40	1356MHZ_LEGIC-PRIME
0x80	1356MHZ_I-CODE-1
0x100	1356MHZ_ISO14443-A
0x200	1356MHZ_PICO-TAG
0x400	1356MHZ_IID-P
0x800	1356MHZ_ISO14443-B
0x01	868_ISO180006C
0x10000000	1356MHZ_ISO15693_addressed_transparent ISO15693 (to be used with 1356MHZ_ISO15693)
0x20000000	1356MHZ_ISO15693_RWblocks (to be used together with 1356MHZ_ISO15693)
0x40000000	1356MHZ_LEGIC file system (to be used together with ISO type above)
0x80000000	1356MHZ_DUALECHO (not compatible with 1356MHZ_ISO14443-A)

Transponder chip types

Following tables contain lists of transponder chip types, supported by c_get_transponder_parameters type detection based on driver versions above 0x10.0x4E.

Code	Transponder chip type
0x04	ISO15693 I-Code SLI
0x07	ISO15693 tag-it
0x16	ISO15693 iID-M
0x05	ISO15693 my-D custom mode
0x41	ISO15693 I-Code UID
0x90	EM4423 / EM echo
0xA0	ISO15693 iID-M2
0xB0	ISO14443-A NXP N-TAG 203
0xB1	ISO14443-A NXP N-TAG 213
0xB2	ISO14443-A NXP N-TAG 215
0xB3	ISO14443-A NXP N-TAG 216
0xB4	ISO14443-A NXP N-TAG I²C 1K
0xB5	ISO14443-A NXP N-TAG I²C 2K
0xC0	ISO14443-A Mifare Ultralight
0xC1	ISO14443-A Mifare Ultralight C (not used)
0xC4	ISO14443-A Mifare Classic 1K
0xC5	ISO14443-A Mifare Classic 4K
0xC6	ISO14443-A Mifare Mini
0xC7	ISO14443-A Mifare Plus 2K
0xC8	ISO14443-A Mifare Plus 4K
0xC9	ISO14443-A Mifare Desfire 2K
0xD4	ISO15693 I-Code SLI-S
0xD5	ISO15693 I-Code SLI-L
0xD6	ISO15693 I-Code SLI-XS
0xD7	ISO15693 I-Code SLI-XL
0xD8	ISO15693 I-Code SLI-X
0xD9	ISO15693 I-Code SLI-X2

Code	Transponder chip type
0xF0	iID-D
0xF1	I-Code 1
0xF2	iID-N
0xF3	iID-G
0xF4	LEGIC Prime
0xF5	ISO15693 my-D ISO mode
0xF6	iID-L
0xF7	Pico-TAG 16k
0xF8	iID-H
0xF9	Pico-TAG 32k
0xFA	ISO14443-B iID-K
0xFB	iID-Q1
0xFC	iID-Q2
0xFD	iID-P