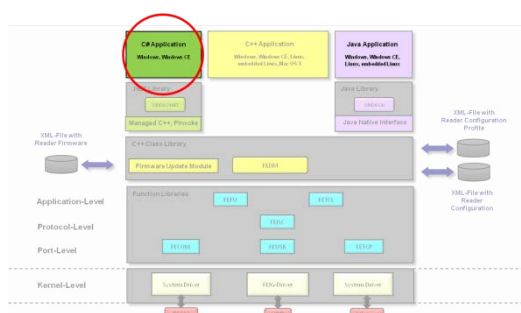


.NET Class Library ID ISC.SDK.NET

Version 4.09.00

Software-Support for OBID i-scan® and OBID® classic-pro



.NET Framework	Target		Operating Systems
	32-Bit (x86)	64-Bit (x64)	
Up from V4.0	X	X	Windows Vista / 7 / 8 / 10 (32- or 64-Bit)
Compact V2.0 and 3.5	X	-	Windows CE 6

Note

© Copyright 2003-2017 by FEIG ELECTRONIC GmbH
Lange Straße 4
D-35781 Weilburg
Germany
eMail: identification-support@feig.de

The indications made in these mounting instructions may be altered without previous notice. With the edition of these instructions, all previous editions become void.

Copying of this document, and giving it to others and the use or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of a utility model or design.

Composition of the information given in these mounting instructions has been done to the best of our knowledge. FEIG ELECTRONIC GmbH does not guarantee the correctness and completeness of the details given and may not be held liable for damages ensuing from incorrect installation.

Since, despite all our efforts, errors may not be completely avoided, we are always grateful for your useful tips.

FEIG ELECTRONIC GmbH assumes no responsibility for the use of any information contained in this manual and makes no representation that they are free of patent infringement. FEIG ELECTRONIC GmbH does not convey any license under its patent rights nor the rights of others.

The installation-information recommended here relate to ideal outside conditions. FEIG ELECTRONIC GmbH does not guarantee the failure-free function of the OBID®-system in outside environment.

OBID® and OBID i-scan® are registered trademarks of FEIG ELECTRONIC GmbH.

Microsoft® .NET is a registered trademark of Microsoft Corporation in the United States and other countries.

Windows is a registered trademark of Microsoft Corporation in the United States and other countries.

Windows Vista is either a registered trademark or trademark of Microsoft Corporation in the United States and/or other countries

I-CODE®, UCODE® and Mifare® are registered Trademarks of NXP Semiconductor

Tag-it™ is a registered trademark of Texas Instruments Incorporated

Electronic Product Code (TM) is a Trademark of EPCglobal Inc.

Jewel (TM) is a trademark of Innovision Research & Technology plc.

Licensing Agreement for use of the software

This is a contract between you and FEIG ELECTRONIC GmbH (hereinafter called "FEIG") concerning the use of the provided software **ID ISC.SDK.NET** (application programs, program libraries, source code examples and documents), hereinafter called licensed material. By installing and using this software you agree to be bound to all the terms and conditions of this Agreement without exception or limitation. If you are not fully or partially in agreement with the terms of this contract, you are prohibited from installing or otherwise using the material(s). The material is property of FEIG and is protected by international copyright law.

§1 Subject and Scope of the Agreement

1. FEIG grants you the right to install the Software and to use it under the conditions specified below.
2. The licensed material is intended for use by an individual developer (single user license). You may install all the components of the licensed material on a hard-disk of a single computer which is intended for your use.
3. Installation and use may also include a network fileserver as long as the use is exclusive to the licensee. A separate license is required for any additional user.
4. You may make a backup copy of the licensed material.
5. FEIG grants you the right to use the documented .NET library OBIDISC4NET.dll, as well as the necessary native libraries, for developing your own application program and to sell this .NET library OBIDISC4NET.dll, as well as the necessary native libraries, only together with your application programs without payment of licensing fees so long as these application programs are used only to control or operate devices and/or systems which are developed and/or sold by FEIG.
6. FEIG grants you the right to use and modify the source code of the supplied program examples for developing your own application programs and to sell these application programs together with the .NET library OBIDISC4NET.dll, as well as the necessary native libraries, without payment of licensing fees so long as these application programs are used only to control or operate devices and/or systems which are developed and/or sold by FEIG.
7. This license material can depend on third-party software. In case of the use of this third-party software the listed license agreements in chapter [Third-party Licensing agreements](#) have to be applied.

§2. Protection of the Licensed Material

1. The licensed material is the intellectual property of FEIG and its suppliers, and its structure, organization and code are the valuable trade secrets of FEIG and its suppliers. The licensed material is also protected by German copyright law, International Treaty provisions and the laws of the country in which it is used.
2. You agree not to modify, adapt, translate, reverse engineer, decompile, disassemble or otherwise attempt to discover the source code of the executable application programs, program libraries and documents.
3. To the extent that FEIG has applied protections such as copyrights and other legal restrictions, you are required to retain these without modification and to incorporate them in unmodified form into all complete or partial copies which you produce.
4. The publication and transmission to third parties of licensed material prohibited as long as no explicit agreement to the contrary has been established between you and FEIG . This provision does not apply to such application programs as have been created and sold under §1 Par. 5 and Par. 6 of this Agreement.

§ 3 Warranty and Limitation of Liability

1. You agree with FEIG that it is not possible to develop electronic data processing programs such that they are without defect for all application conditions. FEIG calls explicit attention to the fact that the installation of a new program may affect already existing software, including software which does not run simultaneous with the new software. In no event will FEIG be liable to you for any consequential, incidental or special damages, including any lost profits or lost savings. If you want to be sure that no already installed program will be affected, you may not install the licensed material.

2. FEIG calls explicit attention to the fact that installing the licensed material may result in irreversible settings and adjustments to devices which may in turn destroy or otherwise make them unusable. FEIG assumes no liability for such actions whether knowingly or unknowingly.

3. No Warranty. The Software is being delivered to you AS IS and FEIG makes no warranty as to its use or performance. FEIG makes no warranties, express or implied, as to noninfringement of third party rights, merchantability or fitness for any particular purpose.

4. FEIG call explicit attention the licensed material is not designed with components and testing for a level of reliability suitable for use in or in connection with surgical implants or as critical components in any life support systems whose failure to perform can reasonably be expected to cause significant injury to a human.

To avoid damage, injury, or death, the user or application designer must take reasonably prudent steps to protect against system failures.

§ 4 Other

1. The parties agree that this constitutes the sole and entire agreement of the parties as to the matter set forth herein and supersedes any previous agreements, understandings, and arrangements between the parties relating hereto. Any modifications or additions must be made in written form.

2. If any provision in this Licensing Agreement should be held illegal or unenforceable by a court having jurisdiction, such provision shall be modified to the extent necessary to render it enforceable without losing its intent, or severed from this Licensing Agreement if no such modification is possible, and other provisions of this Licensing Agreement shall remain in full force and effect.

3. This License Agreement shall be construed, interpreted, and governed by the laws of the Federal Republic of Germany and the venue of any legal action against FEIG shall be Weilburg.

Please refer any questions concerning these Agreements to:

FEIG ELECTRONIC GmbH
Lange Straße 4
35781 Weilburg
Tel.: 06471 / 31090
Fax: 06471 / 310999
E-Mail: obid-support@feig.de
Internet: <http://www.feig.de>

Third-party Licensing agreements

Licensing agreement of openssl organization

The following license issues are to be applied in the case that encrypted data transmission is used.

LICENSE ISSUES =====

The OpenSSL toolkit stays under a dual license, i.e. both the conditions of the OpenSSL License and the original SSLeay license apply to the toolkit. See below for the actual license texts. Actually both licenses are BSD-style Open Source licenses. In case of any license issues related to OpenSSL please contact openssl-core@openssl.org.

OpenSSL License -----

=====
Copyright (c) 1998-2008 The OpenSSL Project. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgment:
"This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit. (<http://www.openssl.org/>)"
4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact openssl-core@openssl.org.
5. Products derived from this software may not be called "OpenSSL" nor may "OpenSSL" appear in their names without prior written permission of the OpenSSL Project.
6. Redistributions of any form whatsoever must retain the following acknowledgment:
"This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (<http://www.openssl.org/>)"

THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE OpenSSL PROJECT OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
=====

This product includes cryptographic software written by Eric Young (eay@cryptsoft.com). This product includes software written by Tim Hudson (tjh@cryptsoft.com).

Original SSLeay License -----

Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com) All rights reserved.

This package is an SSL implementation written by Eric Young (eay@cryptsoft.com).
The implementation was written so as to conform with Netscapes SSL.

This library is free for commercial and non-commercial use as long as the following conditions are adhered to. The following conditions apply to all code found in this distribution, be it the RC4, RSA, lhash, DES, etc., code; not just the SSL code. The SSL documentation included with this distribution is covered by the same copyright terms except that the holder is Tim Hudson (tjh@cryptsoft.com).

Copyright remains Eric Young's, and as such any Copyright notices in the code are not to be removed. If this package is used in a product, Eric Young should be given attribution as the author of the parts of the library used.

This can be in the form of a textual message at program startup or in documentation (online or textual) provided with the package.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. All advertising materials mentioning features or use of this software must display the following acknowledgement:

"This product includes cryptographic software written by Eric Young (eay@cryptsoft.com)"

The word 'cryptographic' can be left out if the routines from the library being used are not cryptographic related :-).

4. If you include any Windows specific code (or a derivative thereof) from the apps directory (application code) you must include an acknowledgement:

"This product includes software written by Tim Hudson (tjh@cryptsoft.com)"

THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The licence and distribution terms for any publically available version or derivative of this code cannot be changed. i.e. this code cannot simply be copied and put under another distribution licence [including the GNU Public Licence.]

Contents:

Third-party Licensing agreements	5
Licensing agreement of openssl organization	5
1. Introduction	14
1.1. Overview of all software modules	15
1.2. Supported operating systems	16
2. Revisions since the previous version	17
3. Installation	18
3.1. Installation on Development Computer	18
3.2. Installation on Target Computer	20
3.2.1. Dependencies of library files for use with 32-Bit Framework V4.0 and higher.....	21
3.2.2. Dependencies of library files for use with 64-Bit Framework V4.0 and higher.....	22
3.3. Control of installed MFC-Version 11.0 (only Win10)	22
3.4. Application Development with 32-Bit Framework for 64-Bit Windows	23
3.5. Supported Development Tools	24
4. Overview of the classes	25
4.1. Reader class FedmlscReader	25
4.2. Table classes FedmlsoTableItem and FedmBrmTableItem	26
4.3. Class FedmlscFunctionUnit	28
4.4. Class FedmlscPeopleCounter	29
4.5. Help classes and interfaces	30
4.5.1. FedmlscReaderInfo.....	30
4.5.2. FedmlscReaderTime.....	30
4.5.3. FedmCprAdu.....	30
4.5.4. FedmCprCommandQueue	30
4.5.5. FeUsb and FeUsbScanSearch.....	31
4.5.6. FeHexConvert.....	31
4.5.7. The FelscListener interface	31
4.5.8. The FelscListenerConst structure	31
4.5.9. The FeUsbListener interface	32
4.5.10. The FeUsbListenerConst interface	32
4.5.11. The FedmTaskListener interface	32
4.5.12. The FedmTaskOption property	32
4.5.13. The Fedm structure.....	32
4.5.14. The FedmlscReaderConst structure	32
4.5.15. The FedmlscReaderID structure	32

4.5.16. The FedmlscFunctionUnitID structure	32
4.6. Exception classes	33
4.6.1. FedmException	33
4.6.2. FePortDriverException	33
4.6.3. FeReaderDriverException	33
5. Basic properties of the reader class	34
5.1. Initializing and finalizing	34
5.1.1. Initializing	34
5.1.2. Finalizing.....	35
5.2. Administering the communications channels.....	35
5.3. Communication with the reader	36
5.3.1. Synchronous communication	36
5.3.2. Asynchronous Communication.....	38
5.3.3. Secured data transmission with encryption	40
5.3.3.1. Overview.....	40
5.3.3.2. Feedback of error cases	40
5.3.3.3. Notes for Programmers.....	41
5.4. Communication with a function unit.....	42
5.5. Communication with a People Counter	43
5.6. Data containers	44
5.6.1. Data exchange.....	44
5.6.1.1. Constant Data.....	44
5.6.1.2. Data type bool.....	44
5.6.1.3. Data type byte.....	44
5.6.1.4. Data type byte[].....	44
5.6.1.5. Data type uint.....	45
5.6.1.6. Data type long.....	45
5.6.1.7. Data type string.....	45
5.6.2. Access constants for temporary protocol data.....	46
5.6.3. Reader Configuration Parameters in the Namespace OBID.ReaderConfig	48
5.6.4. Management of the reader configuration.....	49
5.6.5. Serializing	52
5.7. Tables.....	54
5.8. Communication with Transponders in the Host-Mode	55
6. Error handling	57
6.1. Return value	57
6.2. Exceptions	57
7. Library Reference	58
7.1. FedmlscReader.....	58
7.1.1. GetDependentLibVersions	58
7.1.2. FedmlscReader	59

7.1.3. ConnectCOMM	60
7.1.4. ConnectTCP	61
7.1.5. ConnectUSB	62
7.1.6. DisConnect	63
7.1.7. GetTcpConnectionStatus	63
7.1.8. SetPortPara	64
7.1.9. GetPortPara	66
7.1.10. SetBusAddress	66
7.1.11. GetBusAddress	67
7.1.12. GetFamilyCode	67
7.1.13. GetReaderName	68
7.1.14. GetTransponderName	68
7.1.15. GetReaderType	69
7.1.16. SetReaderType	69
7.1.17. SendProtocol	70
7.1.18. SendTransparent	70
7.1.19. TagInventory	71
7.1.20. TagSelect	72
7.1.21. SendTclApu. SendTclPing, SendTclDeselect	73
7.1.22. CancelTclApu	75
7.1.23. SendCommandQueue	76
7.1.24. SendSAMCommand	77
7.1.25. FindBaudrate	78
7.1.26. Serialize	79
7.1.27. TransferReaderCfgToXmlFile, TransferXmlFileToReaderCfg	79
7.1.28. ReaderAuthentication	80
7.1.29. ReadReaderInfo	81
7.1.30. ReadCompleteConfiguration	81
7.1.31. WriteCompleteConfiguration	81
7.1.32. ResetCompleteConfiguration	82
7.1.33. ApplyConfiguration	82
7.1.34. GetLastError	83
7.1.35. GetLastStatus	83
7.1.36. GetErrorText	83
7.1.37. GetStatusText	83
7.1.38. GetData	84
7.1.39. SetData	84
7.1.40. GetConfigPara	85
7.1.41. SetConfigPara	85
7.1.42. TestConfigPara	86
7.1.43. GetByteContainer	87
7.1.44. SetByteContainer	87
7.1.45. GetTagList	88
7.1.46. GetTagHandler, GetSelectedTagHandler	88
7.1.47. CreateNonAddressedTagHandler	88
7.1.48. GetTableItem	89
7.1.49. SetTableItem	89

7.1.50. GetTable	90
7.1.51. SetTable	90
7.1.52. GetTableSize	91
7.1.53. SetTableSize.....	91
7.1.54. GetTableLength	92
7.1.55. SetTableLength.....	92
7.1.56. ResetTable	93
7.1.57. GetTableData.....	94
7.1.58. SetTableData	95
7.1.59. VerifyTableDataBlocks	96
7.1.60. FindTableIndex	97
7.1.61. AddEventListener.....	98
7.1.62. RemoveEventListener.....	101
7.1.63. StartAsyncTask.....	102
7.1.64. CancelAsyncTask	104
7.1.65. TriggerAsyncTask	104
7.2. FedmIsoTableItem, FedmBrmTableItem	105
7.2.1. Data members in FedmISOTableItem	105
7.2.2. Data members in FedmBRMTableItem	107
7.2.3. GetData	109
7.2.4. SetData.....	110
7.2.5. GetRSSI.....	111
7.2.6. VerifyDataBlocks.....	112
7.2.7. IsDataValid	113
7.2.8. GetIdentifier	113
7.3. FedmIsCFunctionUnit.....	114
7.3.1. FedmIsCFunctionUnit	114
7.3.2. GetFUType	114
7.3.3. GetLastError	115
7.3.4. GetErrorText	115
7.3.5. SendProtocol	116
7.3.6. GetData	117
7.3.7. SetData.....	117
7.3.8. AddChild	118
7.3.9. DeleteChild	118
7.3.10. GetChild.....	118
7.4. FedmIsCPeopleCounter	119
7.4.1. GetCounterValues	119
7.4.2. SetCounterValues.....	120
7.4.3. SetOutputsOn	121
7.4.4. SetOutputsOff	122
7.4.5. SetOutputsFlashing.....	123
7.5. FeHexConvert.....	124
7.5.1. ByteArrayToHexStringWithSpaces.....	124
7.5.2. ByteArrayToHexString	124

7.5.3. ByteToHexString.....	125
7.5.4. IntegerToHexString.....	125
7.5.5. LongToHexString.....	126
7.5.6. HexStringToByte.....	126
7.5.7. HexStringToByteArray.....	127
7.5.8. HexStringToInteger.....	127
7.5.9. HexStringToLong.....	128
7.5.10. isHexString.....	128
7.5.11. GetMemIDOfID.....	128
7.5.12. GetByteCntOfID.....	129
7.5.13. GetAdrOfID.....	129
7.6. FelscListener.....	130
7.6.1. OnSendProtocol.....	130
7.6.2. OnReceiveProtocol.....	130
7.7. FeUsbListener.....	131
7.7.1. OnConnectReader.....	131
7.7.2. OnDisConnectReader.....	131
7.8. FedmTaskListener.....	132
7.8.1. OnNewTag.....	132
7.8.2. OnNewNotification.....	132
7.8.3. OnNewReaderDiagnostic.....	133
7.8.4. OnNewPeopleCounterEvent.....	133
7.8.5. OnNewSAMResponse.....	134
7.8.6. OnNewApduResponse.....	134
7.8.7. OnNewQueueResponse.....	135
7.9. FedmException.....	136
7.10. FedmPortDriverException.....	136
7.11. FedmReaderDriverException.....	136
8. Examples for using the function SendProtocol.....	137
8.1. Basic commands.....	138
8.2. Table oriented commands.....	149
8.2.1. Anomaly of the addressed mode.....	149
8.2.2. Examples for using the ISO table with [0xB0] Commands.....	149
8.2.3. Examples for using the ISO table with [0xB3] Commands.....	160
8.2.4. Commands for Buffered Read Mode.....	162
8.3. Commands for function unit.....	164
9. Example for using TagHandler classes.....	166
10. Appendix.....	168
10.1. List of error codes.....	168

10.2. Supported OBID® Readers	171
10.3. Supported Transponders	172
10.4. TCP Status	173
10.5. List of constants	174
10.5.1. General constants	174
10.5.2. Constants for tableID	174
10.5.3. Constants for dataID	174
10.6. Revision history	181

Remarks concerning the documentation for this library

System manuals for the OBID® Readers actually used must also be referred to for understanding the classes and methods.

FEIG ELECTRONIC GmbH does not duplicate information about OBID® Readers in different manuals or include cross-references to certain page numbers of another document. This is because the manuals are constantly updated, and helps to eliminate mistakes resulting from information obtained from out-of-date documents. We therefore encourage the user of this library to always verify that he is using the current manuals. The newest versions can always be obtained from FEIG ELECTRONIC GmbH.

Important notes:

You may use this library only if you have first agreed to the licensing terms found on the reverse side.

1.Introduction

The .NET class library ID OBIDISC4NET from FEIG ELECTRONIC GmbH represents yet another component for simplifying the development of application programs in .NET Frameworks for OBID *i-scan*® and OBID®*classic-pro* readers.

This manual is intended as an introduction to the library.

The .NET class library ID OBIDISC4NET currently supports Windows.

The .NET class library ID OBIDISC4NET is based on the C++ class library ID FEDM as well as the native function libraries ID FECOM, ID FEUSB, ID FETCP, ID FEISC and ID FEFU. The .NET class library therefore consists only of a wrapper. Nevertheless, the full functionality of the C++ class library is accessible for the .NET Framework:

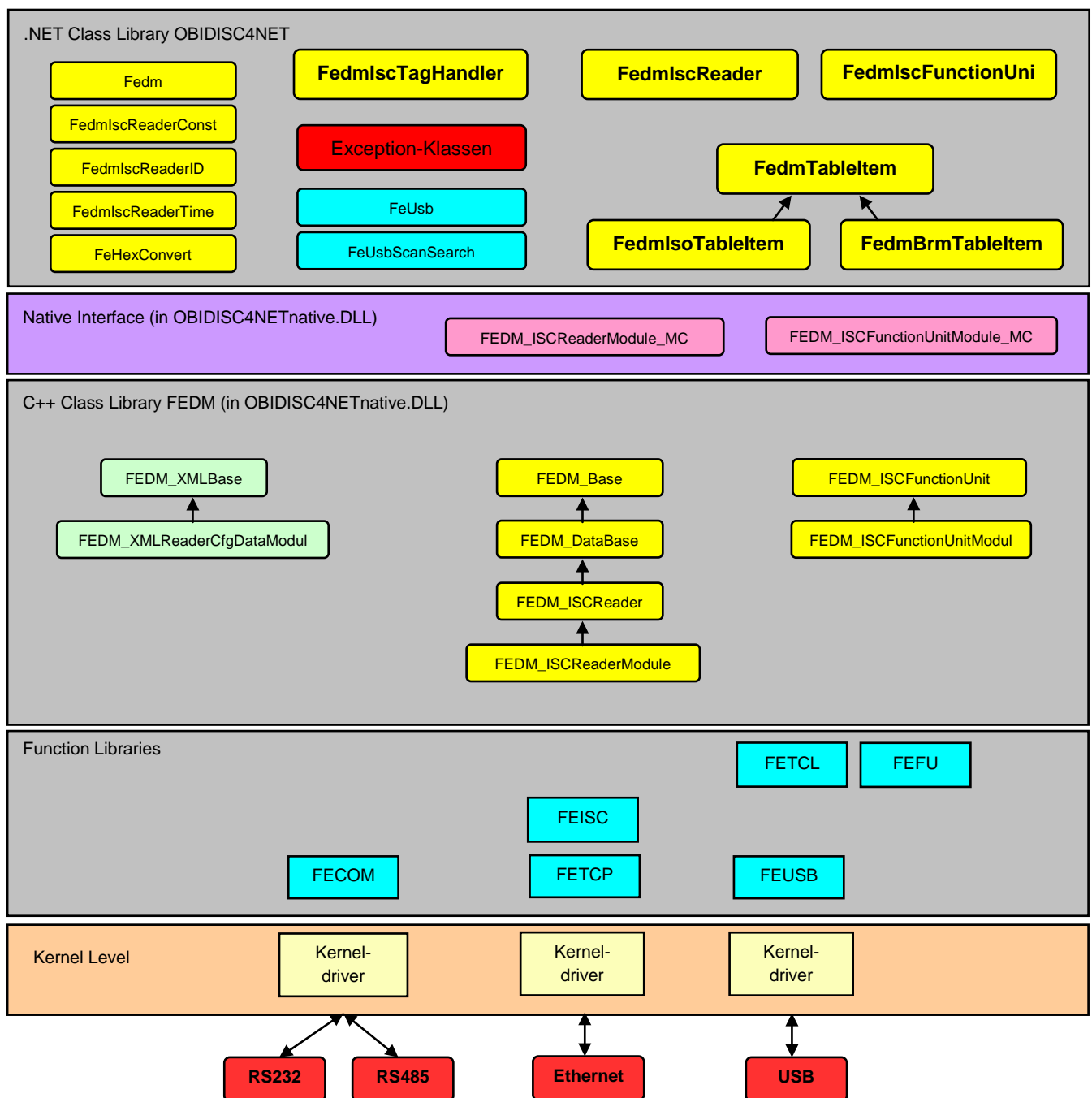
- A uniform organizational principle for savable data from reader and transponder in data containers and tables.
- Overloaded methods for access to the data containers and tables.
- A single, easy to use communications method.
- Synchronous and asynchronous communication
- Complete error handling using exceptions or return values from methods.
- A simple way of serializing reader configuration data in an XML file.

Important note:

The ID OBIDISC4NET class library is being constantly adapted. We make effort to maintain the documented status. Nevertheless, changes cannot be precluded.

1.1. Overview of all software modules

The following illustration shows the individual software modules upon which the ID OBIDISC4NET .NET class library is based. The **FedmlscReader** class is the main class. Through it the communications channel is opened and the entire communication with the reader is carried out on this channel. FedmlscReader builds directly upon the C++ class FEDM_ISCReaderModule, which contains the implementation of the native methods. The managed C++ class FEDM_ISCReaderModule_MC is the mediator between the native and the managed parts.



1.2. Supported operating systems

The following matrix informs about the supported .NET-Framework versions.

.NET Framework	Target		Operating Systems
	32-Bit (x86)	64-Bit (x64)	
Up from V4.0	X	X	Windows Vista / 7 / 8 / 10(32- or 64-Bit) with installed 32- or 64-Bit Framework
Compact V2.0 and 3.5	X	-	Windows CE 6

A version for Windows CE 6 is available in request.

The .NET Micro Framework is not supported.

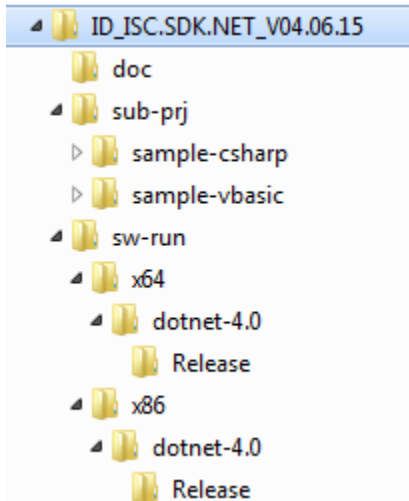
2. Revisions since the previous version

- Updated Reader Configuration namespaces in ReaderConfig
- New **TagHandler** classes for
 - STM ST25DV04K and STM ST25DV16/64K
 - NXP ICODE DNA
 - NXP ICODE SLIX2
- TagHandler class **FedmlscTagHandler _EPC_Class1_Gen2**
 - New communication methods Select, Untraceable, Authenticate, Challenge and ReadBuffer according EPC class1 gen2 standard v2.0.1 and ISO/IEC 29167-x
- TagHandler class **FedmlscTagHandler _ISO15693**
 - New communication methods Authenticate, Challenge and ReadBuffer according ISO/IEC 15693-3 Amd4 and ISO/IEC 29167-x
- Bugfix in TagHandler class **TH_ISO14443_4_MIFARE_Plus_SL3**
 - All Read methods returns data from internal Rx buffer instead of internal Tx buffer.

Please note also the revision history in the Appendix to this document.

3.Installation

Normally, this package is shipped together with other libraries in a Software Development Kit (SDK). Copy the SDK into a directory of your choice.



Picture: Directory structure of the SDK

3.1. Installation on Development Computer

The SDK is shipped with 32-Bit runtime package and is compliant with .NET Framework up from V4.0 (inside directory \sw-run\x86\dotnet-4.0\Release).

The 64-Bit runtime package is compliant with .NET Framework up from V4.0 and can be found in the directory \sw-run\x64\dotnet-4.0\Release.

The library files of all directories should never be mixed!

Files in sw-run\x86\dotnet-4.0\Release	Description
FECOM.DLL	32-Bit native library for serial interface
FETCP.DLL	32-Bit native library for TCP/IP
FEUSB.DLL	32-Bit native library for USB
FEISC.DLL	32-Bit native library for OBID i-scan® and OBID® <i>classic-pro</i> Reader
FETCL.DLL	32-Bit native library for OBID® <i>classic-pro</i> Reader
FEFU.DLL	32-Bit native library for OBID i-scan® Function Units
OBIDISC4NETnative.DLL	32-Bit native library with wrapper layer for Framework V4.0 and higher
OBIDISC4NET.DLL	32-Bit .NET library for Framework V4.0 and higher

Files in sw-run\x64\dotnet-4.0\Release	Description
FECOM.DLL	64-Bit native library for serial interface
FETCP.DLL	64-Bit native library for TCP/IP
FEUSB.DLL	64-Bit native library for USB
FEISC.DLL	64-Bit native library for OBID i-scan® and OBID® <i>classic-pro</i> Reader
FETCL.DLL	64-Bit native library for OBID® <i>classic-pro</i> Reader
FEFU.DLL	64-Bit native library for OBID i-scan® Function Units
OBIDISC4NETnative.DLL	64-Bit native library with wrapper layer for Framework V4.0 and higher
OBIDISC4NET.DLL	64-Bit .NET library for Framework V4.0 and higher

Installation is quite simple:

Copy all DLL files to a working directory. It is not recommended to copy all DLL files to the Windows system directory due to avoid version conflicts with other installations.

Inside your .NET project, you must create a link to the file OBIDISC4NET.DLL.

Note: The Assembly file OBIDISC4NET.DLL is signed with a strong name. This enables to sign applications, depend on this assembly, too to maximize the security of the system.

3.2. Installation on Target Computer

Together with the application files, the runtime files OBIDISC4NET.dll, OBIDISC4NETnative.dll, and the runtime files of the function libraries FECOM, FEUSB, FETCP, FEISC, FETCL and FEFU must be installed on the target computer. For the Windows 10 operating system the following Microsoft Runtime libraries (vc110) needs to be installed or copied: mfc110.dll, msucr110.dll, msvcp110.dll.

It is recommended to keep the library files in the directory of the application. This avoids version conflicts with later installations which also install these library files, but possibly different versions.

3.2.1. Dependencies of library files for use with 32-Bit Framework V4.0 and higher

The following merge modules are necessary (**only Win10**):

MFC Version	Merge Modules
Version 11.0	Microsoft_VC110_MFC_x86.msm Microsoft_VC110_CRT_x86.msm Note: Both Merge-Modules are also available in 64-Bit (x64) versions. The library files of this SDK are not suitable for using with this 64-Bit version of Merge-Modules. However, 32- and 64-Bit Merge-Modules can be installed together in one Windows.

These Merge Modules can be updated on the Development Computer with Windows Update (recommended) and can then be added to a Setup project.

Alternatively, the installation of the Visual C++ Runtime Libraries can be realized with the download site of Microsoft. For each version of MFC you can find a file called vcredist_x86.exe for download.

In both cases, it must be guaranteed to have at least the above listed version number installed. See [3.3. Control of installed MFC-Version 11.0](#) to learn, how to check the installed version numbers.

Link to Microsoft download site Visual C++ 2012 Redistributable Package (x86):

<https://www.microsoft.com/en-us/download/details.aspx?id=30679>

3.2.2. Dependencies of library files for use with 64-Bit Framework V4.0 and higher

The following merge modules are necessary (**only Win10**):

MFC Version	Merge Modules
Version 11.0	Microsoft_VC110_MFC_x64.msm Microsoft_VC110_CRT_x64.msm Note: Both Merge-Modules are also available in 32-Bit (x86) versions. The library files of this SDK are not suitable for using with this 32-Bit version of Merge-Modules. However, 32- and 64-Bit Merge-Modules can be installed together in one Windows.

These Merge Modules can be updated on the Development Computer with Windows Update (recommended) and can then be added to a Setup project.

Alternatively, the installation of the Visual C++ Runtime Libraries can be realized with the download site of Microsoft. For each version of MFC you can find a file called vcredist_x64.exe for download.

In both cases, it must be guaranteed to have at least the above listed version number installed. See [3.3. Control of installed MFC-Version 11.0](#) to learn, how to check the installed version numbers.

Link to Microsoft download site for Visual C++ 2012 Redistributable Package (x64):

<https://www.microsoft.com/en-US/download/details.aspx?id=30679>

3.3. Control of installed MFC-Version 11.0 (only Win10)

The directory C:\Windows\System32¹ or C:\Windows\SysWOW64² contains the Microsoft runtime libraries MFC110.DLL, MSVCP110.DLL and MSVCR110.DLL. The version number can be requested with the file properties dialog.

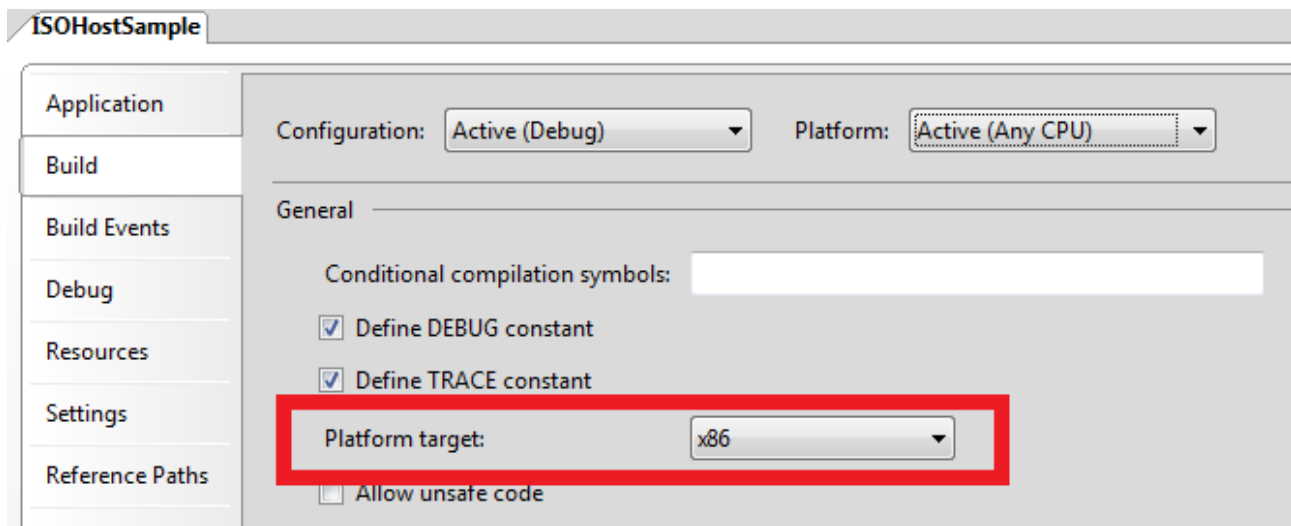
¹ 32-Bit DLLs for 32-Bit Windows resp. 64-Bit DLLs for 64-Bit Windows

² 32-Bit DLLs for 64-Bit Windows

3.4. Application Development with 32-Bit Framework for 64-Bit Windows

The 32-Bit library files of this SDK are only suitable for using together with the 32-Bit (x86) version of the .NET Framework. If an application should run on a 64-Bit Windows, the Platform Target in the project properties must be set to x86.

On the Target PC, the 32-Bit (x86) version of .NET-Framework 4.0 or higher must be installed.



3.5. Supported Development Tools

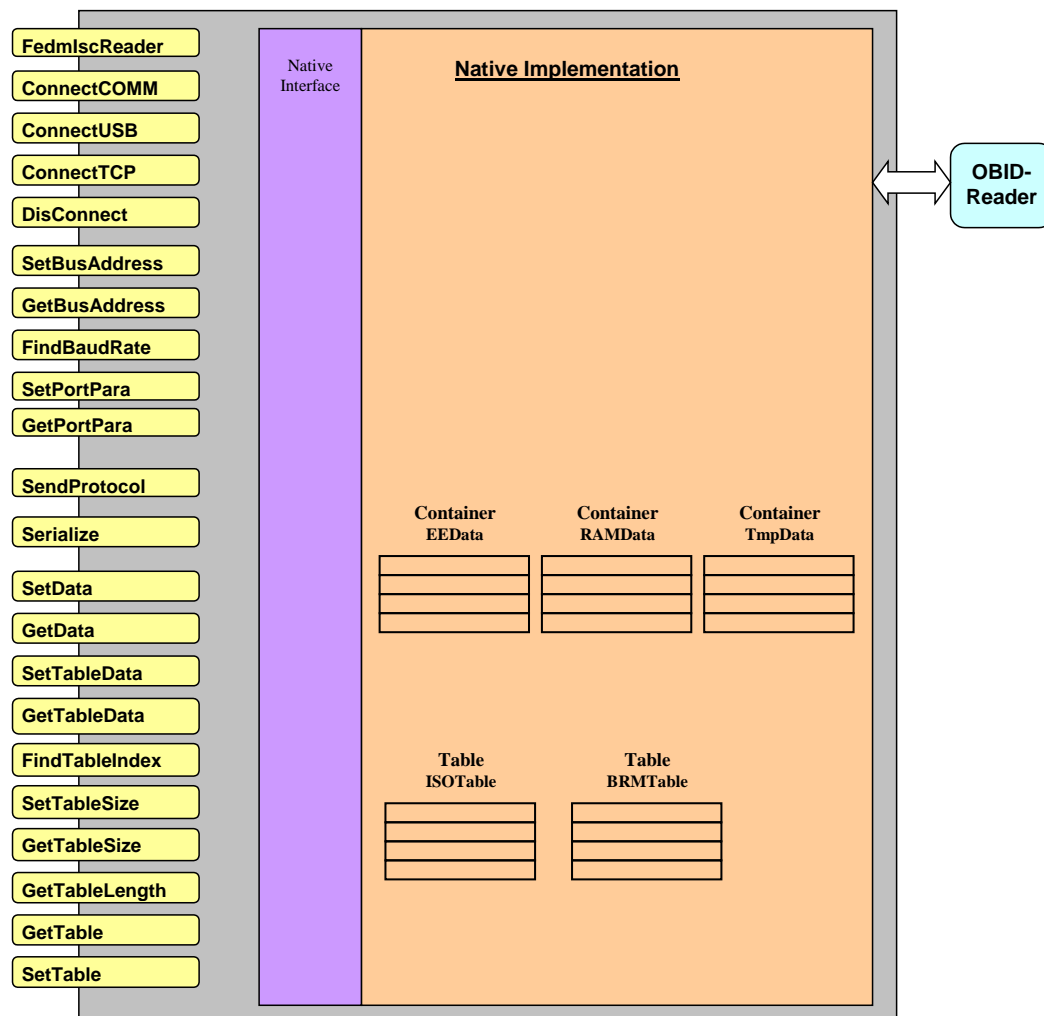
Operating System	IDE	Support
Windows Vista / 7 / 8 / 10	Visual Studio 6 / 2005 / 2008 / 2010	no
	Visual Studio 2012 / 2013 / 2015	yes, beginning with Professional Version
Windows CE	eMbedded Visual C++ 4 / Visual Studio 2005	no
	Visual Studio 2008	yes, beginning with Professional Version

4. Overview of the classes

4.1. Reader class FedmlscReader

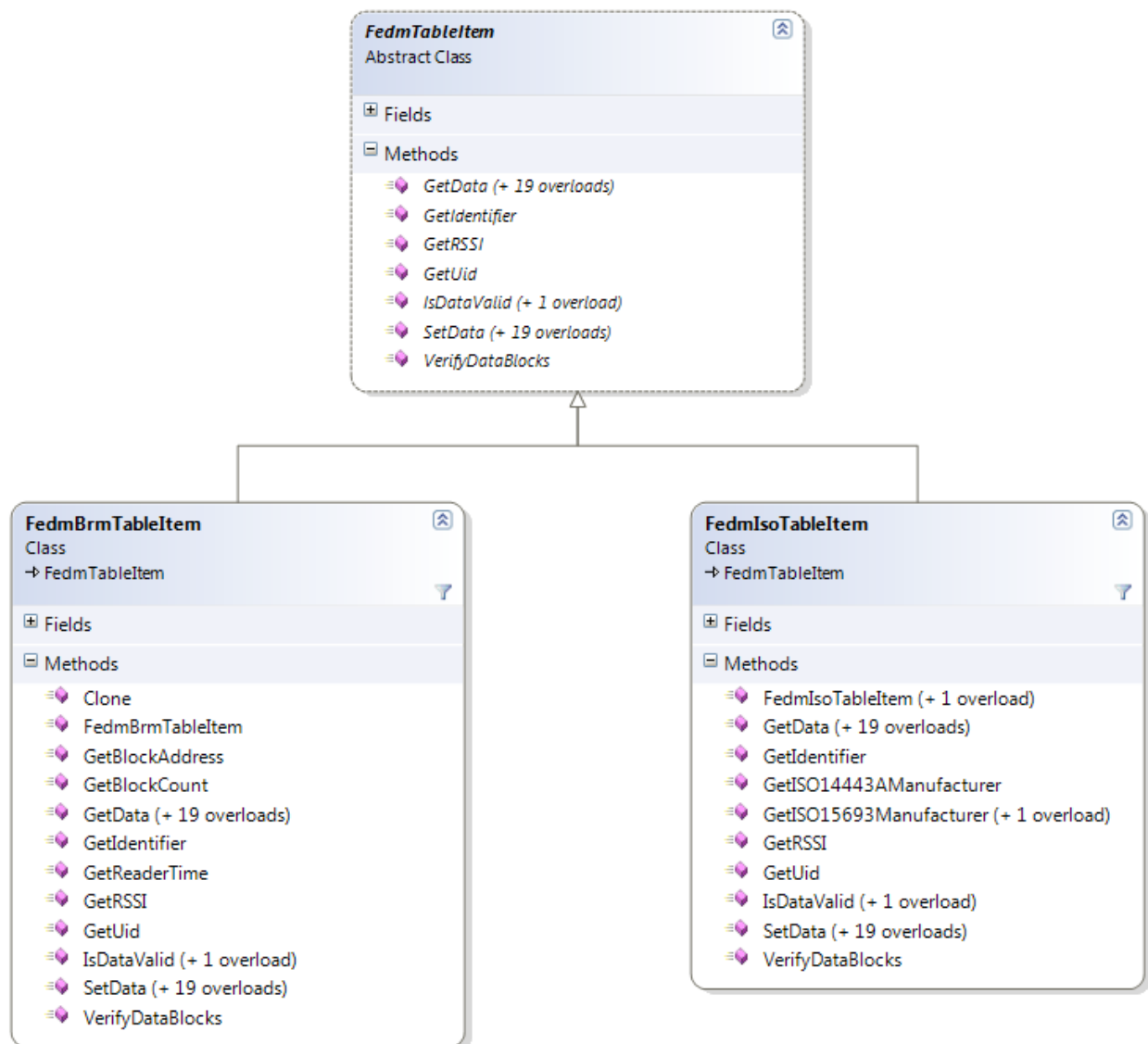
The reader class **FedmlscReader** is the main class of the .NET library. The component diagram shows an overview of the reader class.

Only the most important methods are shown. Attributes are not contained in the class. Refer to the [7.Library Reference](#) for a complete description of the methods.



4.2. Table classes **FedmIsoTableItem** and **FedmBrmTableItem**

The table classes **FedmIsoTableItem** and **FedmBrmTableItem** are derived from the interface **FedmTableItem** and contain transponder data. An array from these classes forms a table, whereby a mixed table is not allowed.



Both classes are an alternative interface to the transponder for the methods *GetTableData* and *SetTableData* of the reader class **FedmIsrReader**. Data can be exchanged with the transponder using only one of the two interfaces.

FedmIsoTableItem contains transponder data that were read with the ISO host mode reader commands or saved there before writing to the transponder.

FedmBrmTableItem contains transponder data that were read by the reader in Buffered Read Mode. Since Buffered Read Mode is purely a read mode, no data can be written in **FedmBrmTableItem** using *SetData*.

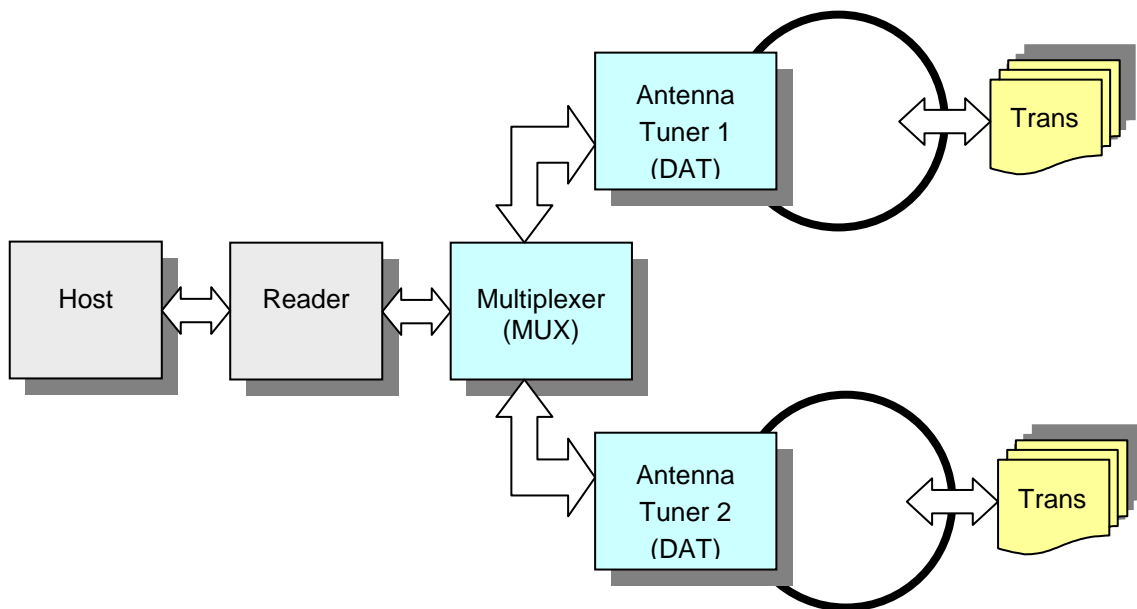
The table classes are always container for read data or data prepared to be written. The data exchange with the Transponder is always executed with the method *SendProtocol* of the reader class **FedmlscReader**.

An easy access to the table elements can be realized with the method *GetTableItem* of the reader class **FedmlscReader** and the direct addressing (since SDK-Version 4.03.00) of the public elements. The use of *GetData* methods are required for table elements like arrays of Transponder data.

4.3. Class FedmlscFunctionUnit

The class **FedmlscFunctionUnit** represents an external function unit (FU) integrated in the antenna cable of the reader. The class has no base class. For a deeper understanding of the possibilities of function units you should read the system manual H30701-xe-ID-B (HF) or H80302-xe-ID-B (UHF). Additional information can be found in the installation guides of the function units.

In consideration of the fact that a function units needs always a reader as a communication bridge, the class **FedmlscFunctionUnit** can only be instantiated if a reader object of type **FedmlscReader** is previously created.



The picture above demonstrates also that external function units are arranged in hierachical order. The function unit class pattern this topology with a list of successors of type **FedmlscFunctionUnit**. Beginning with the first function unit after the reader one can traverse through the tree of function units.

4.4. Class FedmlscPeopleCounter

The class **FedmlscPeopleCounter** represents an external unit connected at the RS485-Bus of theReader. The class has no base class. For a deeper understanding of the possibilities of People Counter you should read the system manual H01011-xe-ID-B. Additional information can be found in the installation guides of the gate antennas.

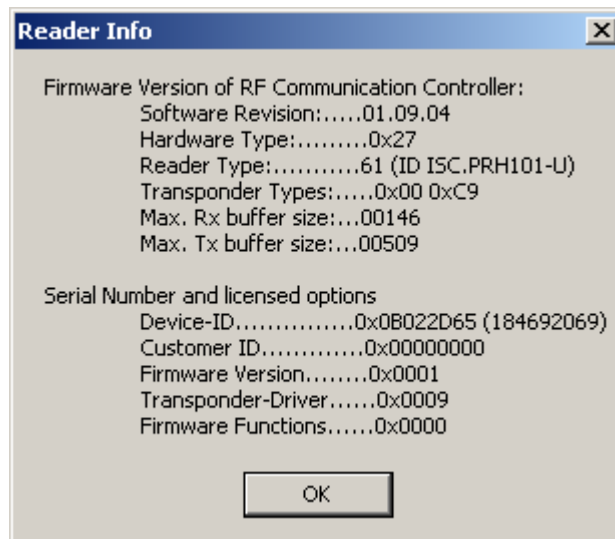
More information can be found in [5.5. Communication with a People Counter](#) and in [7.4. FedmlscPeopleCounter](#).

4.5. Help classes and interfaces

4.5.1. FedmlscReaderInfo

FedmlscReaderInfo is a class that collects all important information of the connected reader after a call of the method *ReadReaderInfo*.

The method *GetReport()* returns a formatted string with all information about the connected reader.



4.5.2. FedmlscReaderTime

FedmlscReaderTime is a class that represents the reader time in Buffered Read Mode.

The object is obtained only using the method *GetReaderTime* of the class **FedmlscBrmTableItem**. The format of date is compliant with ISO 8601.

Example for date: "2012-08-15"

Example for time: "13:01:25.123" (hour:minutes:seconds.milliseconds)

4.5.3. FedmCprAdu

FedmCprAdu is a class supporting the reader class in the asynchronous execution of ISO14443-4 T=CL protocols (APDUs).

4.5.4. FedmCprCommandQueue

FedmCprCommandQueue is a class supporting the reader class in the asynchronous execution of a [0xBC] Command Queue.

4.5.5. FeUsb and FeUsbScanSearch

The class **FeUsb** is a help class for recognizing more than one USB reader when several are connected to the USB at the same time. **FeUsbScanSearch** is a class with search options for a scan procedure on the USB.

If never more than one USB reader is used at a time in your application, you will not need these classes.

FeUsb is a wrapper class for the native library FEUSB.DLL auf. For more informations about FEUSB.DLL please refer to the manual H00501-x-ID-B.

4.5.6. FeHexConvert

The class **FeHexConvert** contains useful methods for converting data.

4.5.7. The FelscListener interface

The **FelscListener** interface enables event handling from the native library. This interface can be used to easily implement a log window for reader logs.

4.5.8. The FelscListenerConst structure

The **FelscListenerConst** structure gathers general constanst for the reader class **FedmlscReader**.

4.5.9. The FeUsbListener interface

The **FeUsbListener** interface enables event handling from the native library. This interface can be used for signaling the connection and disconnection of USB readers.

4.5.10. The FeUsbListenerConst interface

The **FeUsbListenerConst** structure gathers general constants for the usb class **FeUsb**.

4.5.11. The FedmTaskListener interface

The **FedmTaskListener** interface enables event handling from the native library. The different methods signals the application that the transponder and reader data are stored in the internal tables and buffers and ready to be read.

4.5.12. The FedmTaskOption property

The class **FedmTaskOption** contains settings for asynchronous tasks.

4.5.13. The Fedm structure

The **Fedm** structure gathers general constants for the class library.

4.5.14. The FedmlscReaderConst structure

The **FedmlscReaderConst** structure gathers general constants for the reader class **FedmlscReader**.

4.5.15. The FedmlscReaderID structure

The **FedmlscReaderID** gathers all constants for temporary protocol actions for the OBID *i-scan*® and OBID® *classic-pro* readers.

4.5.16. The FedmlscFunctionUnitID structure

The **FedmlscFunctionUnitID** gathers all constants for the OBID *i-scan*® function units.

4.6. Exception classes

4.6.1. FedmException

FedmException is a class which is triggered in exception situations in the area of the native C++ class library FEDM.

4.6.2. FePortDriverException

FePortDriverException is a class which is triggered in exception situations in the area of the native function libraries FECOM, FEUSB und FETCP.

4.6.3. FeReaderDriverException

FeReaderDriverException is a class which is triggered in exception situations in the area of the native function library FEISC.

5. Basic properties of the reader class

The reader class and function unit class methods can be roughly into five categories:

- a) Methods for initializing and finalizing
- b) Methods for the communications channels
- c) Methods for the communication
- d) Methods for data containers and serializing
- e) Methods for tables

5.1. Initializing and finalizing

5.1.1. Initializing

Before using the reader class for the first time, several initializations must be performed:

1. Bus address

The bus address of the reader is preset in the class to 255. Any other address is set using the method *SetBusaddress*. This setting makes sense only for the serial port. Setting of bus address in the library has no affect for the reader setting.

The address for a function unit inside the class is set using the function *SetData*(FedmIscFunctionUnitID.FEDM_ISC_FU_TMP_DAT_ADR, (byte)1)
2. Table size

The tables ISOTable and and BRMTable contained in the reader class **FedmIscReader** do not have a preset size. Therefore you **must** (!) use the method *SetTableSize* to dimension the required table before first communaction with a transponder.

The reference for the size of a table is the maximum number of transponders that will be located in the reader's antenna field at one time.

In general you size only one table, since the reader can not work simultaneously in Buffered Read Mode and ISO-Host Mode.

The following memory capacity per table item is reserved for the tables:

 - BRMTable: 1104 Bytes
 - ISOTable: 17496 Bytes

3. Reader type The reader type must be set in the reader class with one of three options:
1. Automatic (recommended): After a successful connection with one of the methods *ConnectCOMM(..., true)*, *ConnectUSB* or *ConnectTCP* the method *ReadReaderInfo* is executed internally and the reader type is set.
 2. Manually 1: The call of the method *ReadReaderInfo* after a successful opening of a serial port with *ConnectCOMM(..., false)*.
 3. Manually 2: Set of reader type with the method *SetReaderType*. The constants of all reader types are listed in the interface *FedmlscReaderConst*.

5.1.2. Finalizing

In .NET the garbage collector assumes the task of removing no longer needed objects. This works wonderfully in pure .NET applications. But objects that were created in native environment are not under the control of the garbage collector. Thus, before an application is finished the communication ports should be closed with the method *Disconnect*.

5.2. Administering the communications channels

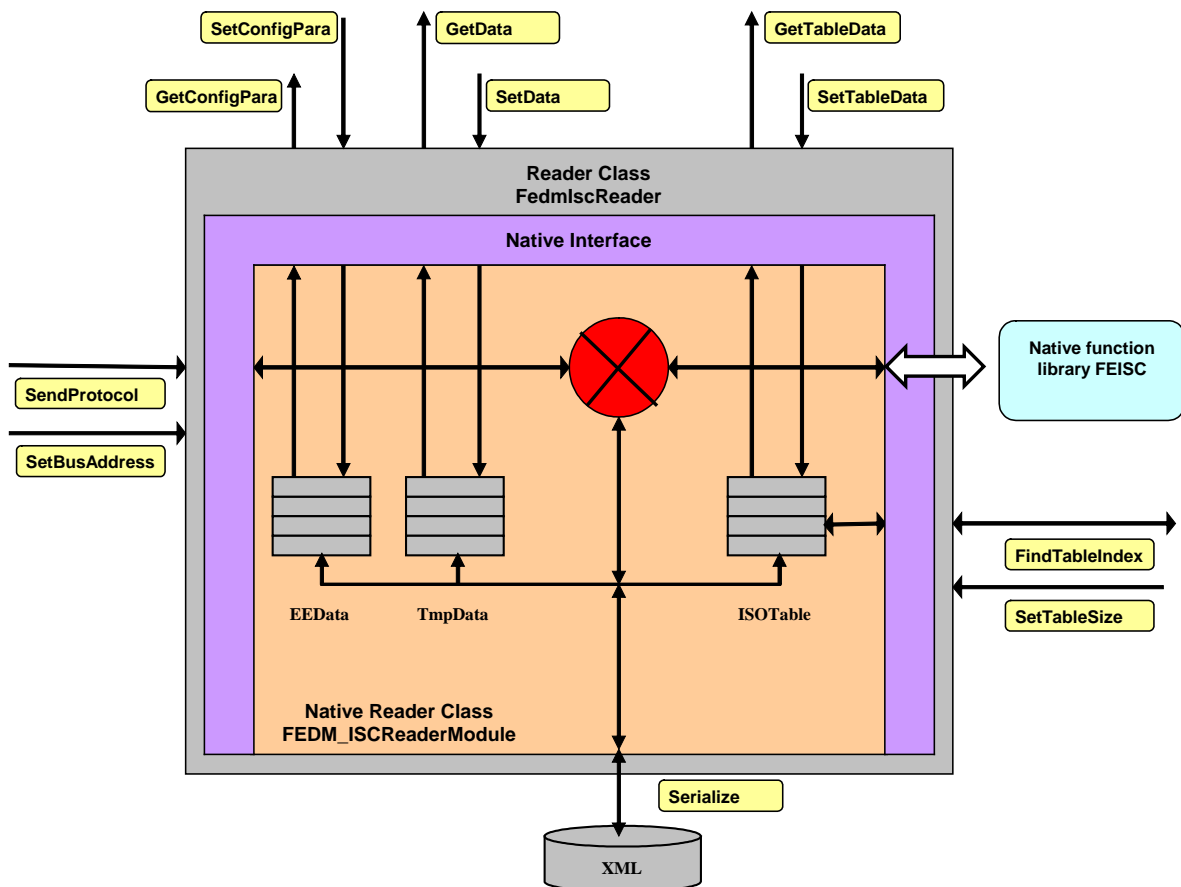
Within the class library there are, with one exception, no classes for the communications channels. Instead methods are integrated in the reader class *FedmlscReader*: *ConnectCOMM*, *ConnectUSB*, *ConnectTCP* open one channel respectively to the reader. *Disconnect* is used to close this channel. For the serial port there is also the method *FindBaudrate*, which detects a reader and correctly configures the port for the communication parameters (baud rate, frame).

In the exceptional case that multiple USB readers have to be supported at the same time in an application, there is the class *FeUsb*, which provides special methods for this case.

5.3.Communication with the reader

5.3.1. Synchronous communication

The synchronous communication sequence in the reader class **FedmlscReader**, which is initiated by a host application, can be explained nicely in the following illustration: In the vertical dimension are the data flows that are moved using the (overloaded) *GetData* respectively *GetConfigPara* and *SetData* respectively *SetConfigPara*, as well as *GetTableData* und *SetTableData*. In addition, the method *Serialize* is sued to enable data flow between a reader object and a file.



In the horizontal axis is the control flow triggered by the method *SendProtocol*, the only communications method. This autonomously and internally gets all the necessary data from the integrated containers before outputting the send protocol and saves the received protocol data there. This means that the application program must write **all** the data needed for this protocol to the corresponding data containers and in the right locations **before** invoking *SendProtocol*. Likewise the receive data are stored at particular locations in corresponding data containers.

The key to the protocol data are so-called access constants for temporary protocol data in the namespace **OBID.ReaderCommand** (e.g. `OBID.ReaderCommand._0x6A.Req.RF_OUTPUT`) and the namespace **OBID.ReaderConfig** for reader configuration parameters (e.g. `OBID.ReaderConfig.OperatingMode.Mode`). Anywhere from a few dozen to a hundred constants and names in the namespace **OBID.ReaderConfig** can be defined for each reader class. The

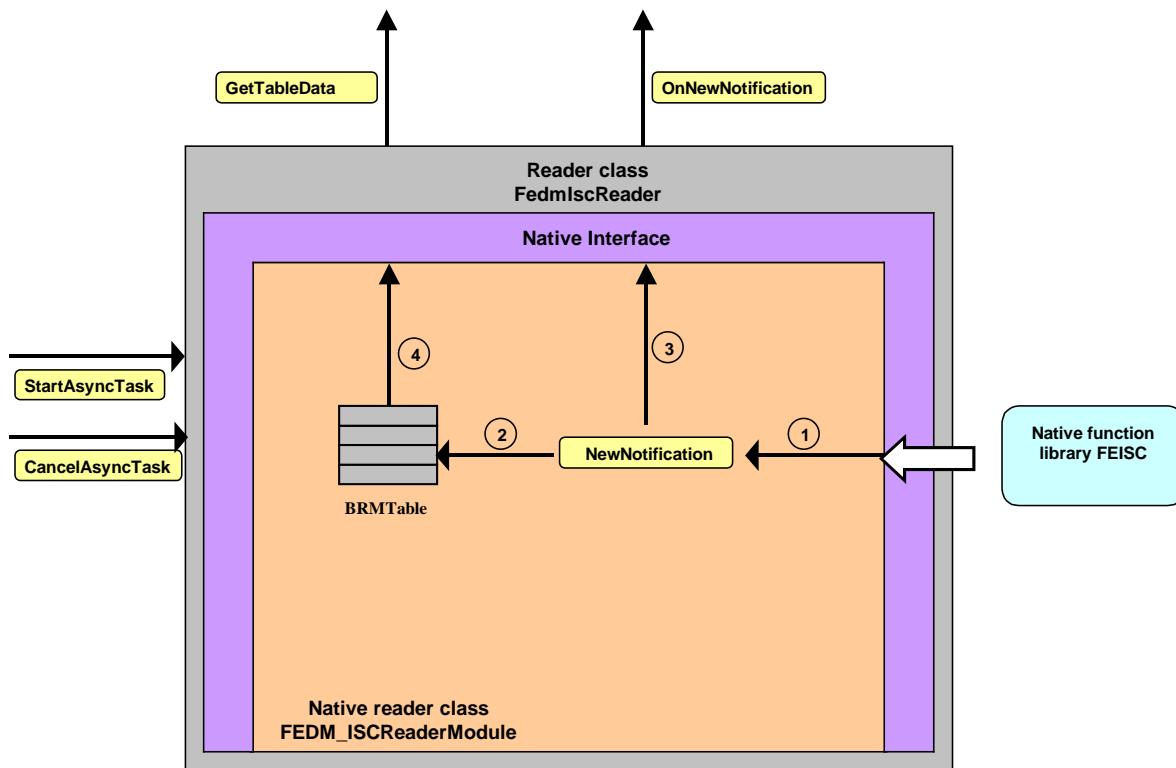
structure is the same for all reader classes and is especially significant. This is explained in detail in [5.6.2. Access constants for temporary protocol data](#) and [5.6.3. Reader Configuration Parameters in the Namespace OBID.ReaderConfig](#). Since the access constants are of key significance for the entire function of the reader class, they are described in detail together with their use in section [8. Examples for using the function SendProtocol](#). The definition of each reader configuration parameter in the namespace **OBID.ReaderConfig** is documented in the system manual of the reader.

The OBID® -Readers on the serial port are bus-compatible and require the bus address in the protocol. This should be set using the method *SetBusAddress*.

5.3.2. Asynchronous Communication

The asynchronous communication is initiated by the method **StartAsyncTask** of the reader class **FedmlscReader** and is triggered by notification events of the reader. Asynchronous tasks can only be used if the reader supports the Notification Mode or the asynchronous option for the Inventory command in the Host Mode³. For each instance of **FedmlscReader** only one asynchronous task can be started.

The information flow can be explained nicely in the following illustration:



In the first step the notification is sent to the native part of the library. In the second step the transponder data are written into the table and the event method of the application is invoked (3rd step). Inside the event method (4th step) the application can use the overloaded methods **getXXTableData** to query the information.

Transponder data from a reader working in Notification Mode will be written into the **BRMTable**. If the reader works in Host Mode the data are written into the **ISOTable**.

³ The latter is only realized in the OBID® *classic-pro* Reader family

The table below lists the assignments of each listener methods to a task:

Task	Task-ID (FedmTaskOption)	Start Method	Listener Method (FedmTaskListener)
Single Inventory	ID_FIRST_NEW_TAG	StartAsyncTask	OnNewTag
Repetitive Inventory	ID_EVERY_NEW_TAG	StartAsyncTask	OnNewTag
Notification	ID_NOTIFICATION	StartAsyncTask	OnNewNotification or OnNewReaderDiagnostic or OnNewPeopleCounterEvent
SAM communication	-	SendSAMCommand	OnNewSAMResponse
Queue command	-	SendQueueCommand	OnNewQueueResponse
T=CL APDU	-	SendTclApdu	OnNewApduResponse
Access Event (FedmIsCMyAxxessReader)	-	StartEventHandler	OnNewMaxAccessEvent or OnNewMaxKeepAliveEvent

5.3.3.Secured data transmission with encryption

5.3.3.1.Overview

Some OBID *i-scan*®- and OBID®*classic-pro* Reader can secure the data transmission with a 256 bit AES algorithm. The Authentication Key (Password) is stored in the Reader and cannot read back. The crypto mode is disabled by default.

The encrypted data transmission is realized with functions of the Open-Source organisation openssl (<http://www.openssl.org>), which are part of the library file libeay32.dll (Windows) resp. libcrypto.so (Linux). The binding to the openssl library file will be affected at runtime with the first call of an openssl function. This has the advantage that all applications are freed from the installation of the openssl library file if no encrypted data transmission is used. In the case that encrypted data transmission is used the license issues of openssl have to be considered.

The encrypted data transmission will be enabled by activating the crypto mode in the Reader configuration with a following CPU-Reset. After that, the Reader accepts only enciphered protocols. To get access rights in crypto mode, the first step must be the establishment of a secured connection with `FedmlscReader.ConnectTCP`, transporting the enciphered password (password contains only nulls by default), to open a new session. Every successive protocol will then enciphered automatically.

Note: After the first authentication a new password should be saved in the Reader and a new authentication with the new password should be executed. This procedure – to switch into the crypto mode first and to change the password secondly – ensures that the new password will be transmitted enciphered! Otherwise the new password will be transmitted plain.

5.3.3.2.Feedback of error cases

A Reader with activated crypto mode ignores all plain protocols and returns the status 0x19 (Crypto Processing Error).

A Reader in plain mode ignores all enciphered protocols and returns the status 0x82 (Command not available).

An authentication into the Reader with a false password will be returned with status 0x12 (Authent Error).

A Reader with activated crypto mode signals with status 0x19 (Crypto Processing Error) an error case in the enciphered transmission. The Host must execute an authentication into the Reader again.

The error code -4093 or -4094 returned by `FedmlscReader.SendProtocol` signals a Host-side error case in the enciphered transmission. The Host must execute an authentication into the Reader again

The error code -4090 signals an error while loading the openssl library file. Probably the library file is not installed or an incompatible version is installed.

5.3.3.3. Notes for Programmers

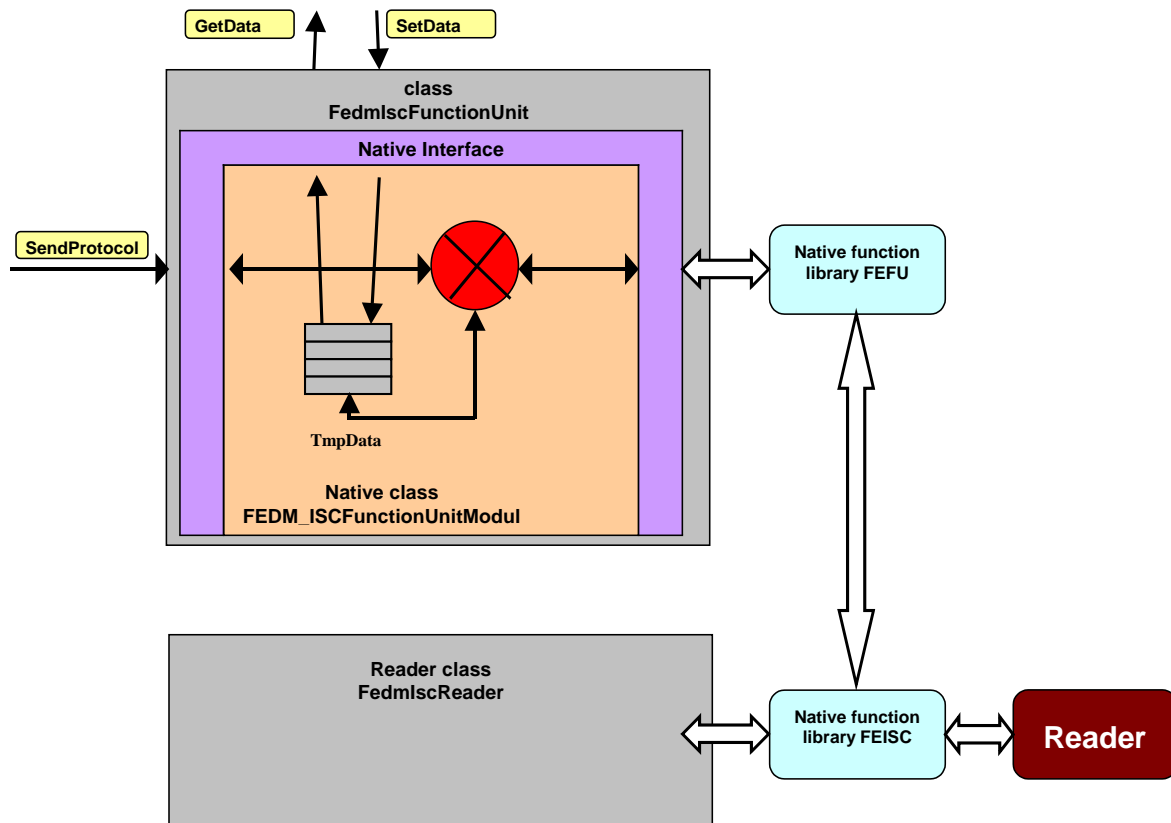
Adding enciphered data transmission into a project needs only few aspects to be considered:

1. Every communication function of class `FedmlscReader` beginning with `Send...` is prepared for plain and enciphered data transmission.
2. It is a requirement to link each OBID *i-scan*®- or OBID® *classic-pro* Reader with one Reader object exclusively, because every Reader object manages the individual session data.
3. Execution of a connection with `FedmlscReader.ConnectTCPwithauthentication` password is required.
4. If the Host application receives after a plain or enciphered data transmission the status 0x19 authentication into the Reader is required.
5. If the error code -4093 or -4094 occurs in the Host application authentication into the Reader is required.
6. In the Notification- and Access-Mode the data transmission is enciphered if the crypto mode is enabled in the Reader. Thus, the password must be added to the class `FedmTaskOption`.
7. If the crypto mode is disabled in the Reader configuration by a configuration protocol, the Reader object changes automatically back into the plain mode with the next plain protocol. This has the advantage that the existing Reader object can be maintained. A new connection is also not necessary.

5.4. Communication with a function unit

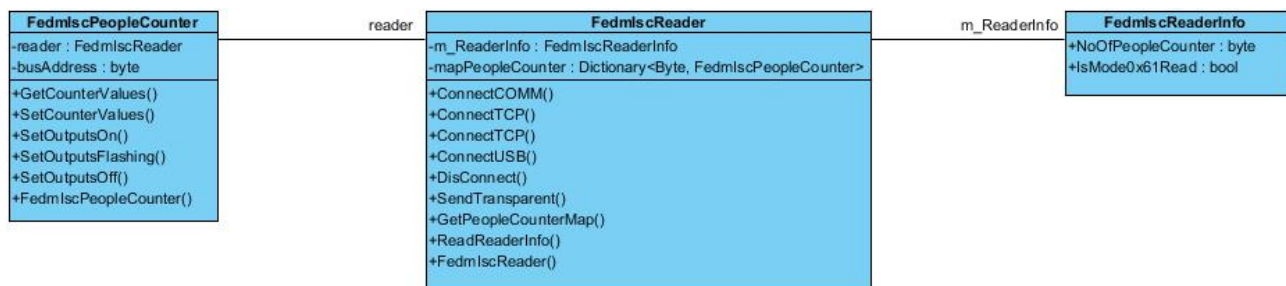
The operation of the communication with a function unit is analog to the communication with a reader. This means that the application program must write **all** the data needed for this protocol to the data container `TmpData` and in the right locations **before** invoking `SendProtocol`. Likewise the receive data are stored at particular locations in data container `TmpData`.

The key to the protocol data are so-called access constants.



5.5. Communication with a People Counter

A Reader detects all People Counters automatically after power-up. From host side, the information about the number of connected People Counters can be queried with the protocol [0x66] Get Reader Info with Mode-Byte 0x61 or with the method `ReadReaderInfo()` of the reader class `FedmlscReader`. Accordingly, all People Counter objects of type `FedmlscPeopleCounter`, collected in a dictionary, can be retrieved with the method `GetPeopleCounterMap()`. The key of the dictionary is the bus address of the People Counter in the range 1...3.



Normally, the method `ConnectUSB`, `ConnectTCP` and (optional) `ConnectCOM` executes after a successful connection internally a `ReadReaderInfo()`. With it, all necessary information about the connected Reader and People Counter are stored in the reader class `FedmlscReader`. The dictionary with connected People Counter objects is already built and can be queried with `GetPeopleCounterMap()` at once. The use of the class `FedmlscPeopleCounter` is explained in detail and with examples in the class reference [7.4. FedmlscPeopleCounter](#).

5.6. Data containers

The task of the data containers is to administer all the reader parameters and temporary protocol data in a structured manner. Internally all data containers are organized as byte arrays in Motorola format (Big Endian). This format is compatible with any OBID®-Reader. Conversion into Intel format required for Intel-based PC's (Little Endian) is handled by the overloaded access methods.

The byte-arrays are organized in 16 or 32-byte blocks. This organization also corresponds to that of the readers.

A total of 3 data containers are integrated.

Data container	Description
EEData	for configuration parameters of the reader
RAMData	for temporary configuration parameters of the reader
TmpData	for general temporary protocol data

5.6.1. Data exchange

Access to the data is possible primarily using the overloaded methods *SetData* and *GetData*. Each method invocation can read or write exactly one parameter, which is identified by an access constant (see. [5.6.2. Access constants](#)).

The following section shows the use of *GetData* and *SetData* for various data types.

5.6.1.1. Constant Data

```
int iErr = SetData(OBID.ReaderCommand._0x80.Req.CfgAdr.LOCATION, false);           // bool
int iErr = SetData(OBID.ReaderCommand._0x66.Req.MODE, (byte)1);                  // byte
int iErr = SetData(OBID.ReaderCommand._0x66.Req.MODE, (long)134);                // long
int iErr = SetData(OBID.ReaderCommand._0x66.Req.MODE, "0134");                  // String
```

5.6.1.2. Data type bool

```
bool data = false;
int iErr = GetData(OBID.ReaderCommand._0x74.Rsp.Inputs.IN1, out data);
iErr = SetData(OBID.ReaderCommand._0xB0.SubCmd._0x01.Req.Mode.MORE, data);
```

5.6.1.3. Data type byte

```
byte data = 1;
int iErr = GetData(OBID.ReaderCommand._0x74.Rsp.INPUTS, out data);
iErr = SetData(OBID.ReaderCommand._0x66.Req.MODE, data);
```

5.6.1.4. Data type byte[]

```
byte[] data = new byte[31];
int iErr = GetData(OBID.ReaderCommand._0x66.Rsp.READER_INFO, out data);
iErr = SetData(OBID.ReaderCommand._0xA0.Req.PASSWORD, data);
```

5.6.1.5. Data type uint

```
uint data = 0;
int iErr = GetData(OBID.ReaderCommand._0xB0.SubCmd._0x24.Rsp.NormAddrMode.DB_ADDRESS_ERROR, out
data);
iErr = SetData(OBID.ReaderCommand._0xB0.SubCmd._0x24.Req.ExtAddrMode.DB_ADR, data);
```

5.6.1.6. Data type long

```
long data = 0;
int iErr = GetData(OBID.ReaderCommand._0x74.Rsp.INPUTS, out data);
iErr = SetData(OBID.ReaderCommand._0x66.Req.MODE, data);
```

5.6.1.7. Data type string

ALL Data that are read using a *GetData(string id, out string data)* method are hex strings. This means for example that the numerical value 159 is not passed as "159" but rather as "9F". String values thus always consist of an even number of characters. The method collection in the class **FeHexConvert** (s. [4.5.6. FeHexConvert](#)) is provided for converting string values into other data types or the reverse.

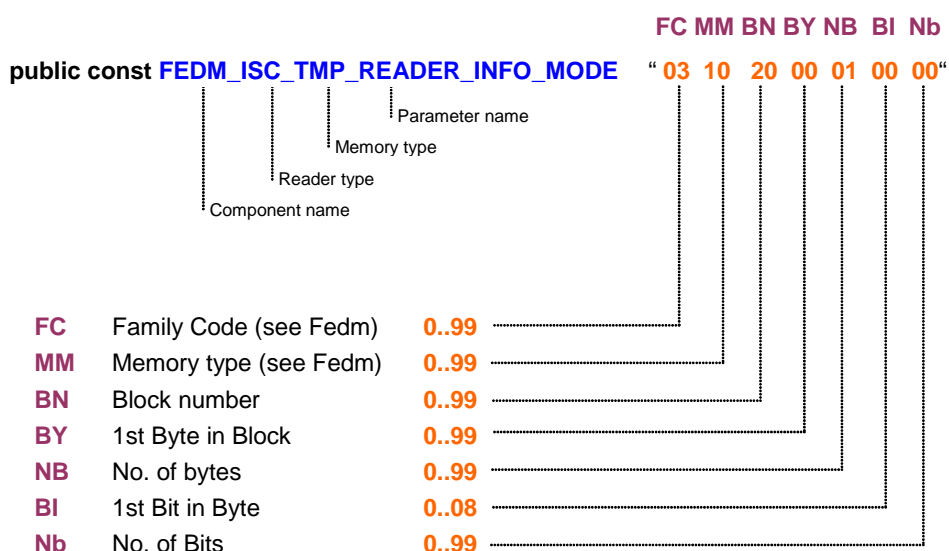
To convert numerical values into string, which in the above example make up the number „159“, the .Net library methods are recommended.

```
String data;
int iErr = GetData(OBID.ReaderCommand._0x74.Rsp.INPUTS, out data);
iErr = SetData(OBID.ReaderCommand._0x66.Req.MODE, data);
```

5.6.2. Access constants for temporary protocol data

The access constants play a central role in data traffic between the application program and data containers of the reader class, as well as within the reader class between protocol method and data container. They identify the parameter and at the same time contain the coded storage location in one of the data containers.

An access constant is a string which generally has the following structure:



These access constants are used exclusively with the methods *SetData* and *GetData*. The access constant says nothing about the data type of a parameter. This is determined only by the data type of the access method. One can therefore read the bus address in the above example either as an integer or as a string or some other plausible data type (see [5.6.1. Data exchange](#)).

All access constants are contained in the structures **FedmlscReaderID** and **FedmlscFunctionUnitID**.

In the course of time, the number of access constants exceeded extremely and the overview got lost. As an alternative since V4.04.00, the namespace **OBID.ReaderCommand** is introduced. It collects all access constants in a structured manner. Examples can be found in the chapter [5.6.1.1](#) to [5.6.1.7](#).

The arrangement in groups follows the principle:

<Namespace>.<CommandByte>.Req.<Parameter> for an parameter in the request protocol

or

<Namespace>.<CommandByte>.Rsp.<Parameter> for an parameter in the responded protocol

Parameters are always in capital letters, groups in mixed letter form. Using Intellisense while coding, the available groups and parameters will be displayed for selecting.

Example:

```
// command byte for inventory
```

```
SetData(ReaderCommand.
```

▲ 1 of 6 ▼ int FedmIsrReader.SetData(string id, bool data)

```
if(bAll)
```

{

SetData(FedmIscRea

```
SetData(FedmIscRea
```

ResetTable(FedmIsc










}

```
else
```

{

SetData(FedmIscRea

}

 _0x22
 _0x31
 _0x34
 _0x64
 _0x66
 _0x6A
 _0x6D
 _0x6E
 0x71

TMP_B0_MODE
 TMP_B0_MODE
 _TABLE);

 TMP_B0_MODE

```
// command byte for inventory
```

```
SetData(ReaderCommand, 0x22,
```

```
//SetData(FedmIsrReaderID.FE... byte)0x01);
```

```
if(bAll)
```

f

```
SetData(FedmIscReaderID.
```

Equals	byte)0x01);
ReferenceEquals	
Req	struct OBID.ReaderCommand_0x22.Req
Rsp	, ucMode);

```
// command byte for inventory
```

```
SetData(ReaderCommand._0x22.Req.
```

```
//SetData(FedmIscReaderID.FEDM_I
```

```
if(bAll)
```

{

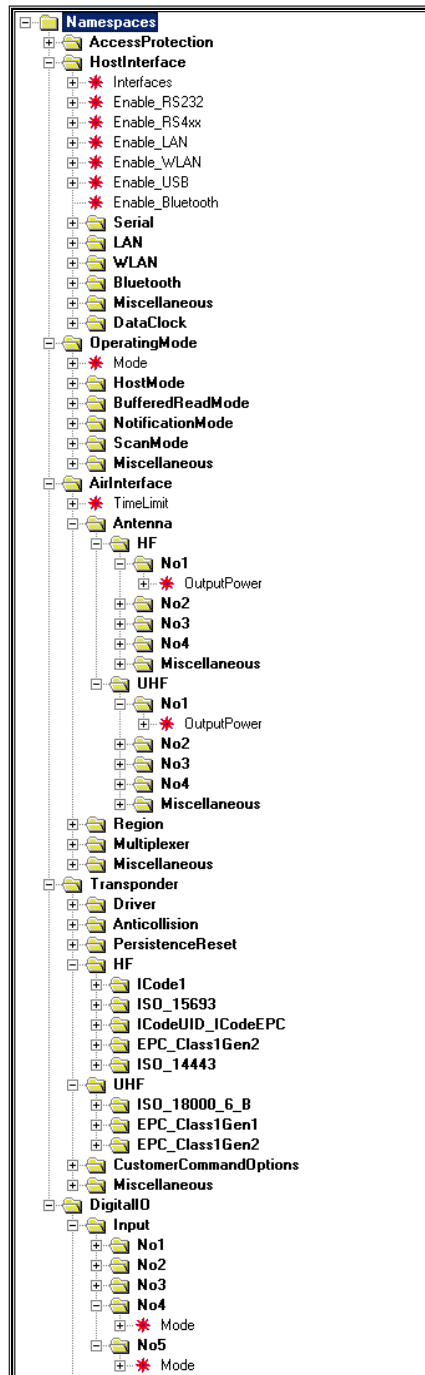
DATA_SETS string Req.DATA_SETS

Equals

ReferenceEquals

5.6.3. Reader Configuration Parameters in the Namespace OBID.ReaderConfig

The data exchange between an application and the data container for reader configuration parameters in the reader class is realized with overloaded methods which passes a string from the namespace OBID.ReaderConfig representing the name of the configuration parameter. All names of reader configuration parameters of all OBID® readers are unified and divided in hierarchical order in groups and subgroups separated by a colon.



Detail of the tree order of the namespace OBID.ReaderConfig

5.6.4. Management of the reader configuration

Each OBID i-scan® and OBID®*classic-pro* reader are controlled by parameters which are stored grouped in blocks in an EEPROM and are described in detail in the system manual for the respective reader. After switching on or resetting the reader, all parameters are loaded into RAM, evaluated and incorporated in the controller.

All parameters can be modified using a protocol so that the behaviour of the reader can be adapted to the application. Ideally, the program ISOStart is used for this adaptation and normally no parameters have to be changed in the application. Despite this, it can happen that one or more parameters from a program have to be changed. This chapter should familiarise you with the procedure using the reader class as an example.

A common characteristic of all readers is the grouping in blocks of thematically related parameters to 14 bytes per configuration block. Each parameter cannot be addressed individually but must always be retrieved together with a configuration block using the protocol [0x80] Read Configuration, then modified and finally written back to the reader with the protocol [0x81] Write Configuration. This cycle must always be complied with and is also checked by the reader class `FedMlscReader`. This means that writing a configuration block without previously reading the same block is not possible.

The reader class manages the configuration data in a (public) byte array `EEData` for data from the EEPROM and `RAMData` for data from the RAM of the reader. The differentiation is important as changes in RAM are used immediately while changes in the EEPROM of the reader do not become active until after a reset. Therefore the reader class has its own byte arrays for both configuration sets.

Using the example of the configuration block CFG2 of the reader ID ISC.LR2000 which contains parameters for the configuration of the digital inputs and outputs, the following should explain how you specifically modify a parameter using the reader class `FedMlscReader`.

Byte	0	1	2	3	4	5	6
Contents	IDLE-MODE		FLASH-IDLE		IN-ACTIVE	0x00	REL1-TIME
Default	0x88A8		0xCC00		0x00		0x00

Byte	7	8	9	10	11	12	13
Contents	REL1-TIME	OUT1-TIME		REL2-TIME		REL3-TIME	REL4-TIME
Default	0x00	0x0000		0x0000			0x0000

IDLE-MODE:

Defines the status of the signal emitters (OUT1 and RELx) during the idle mode.

Bit:	15	14	13	12	11	10	9	8
Function:	REL1 mode		0	0	OUT1 mode		0	0

	7	6	5	4	3	2	1	0
	REL2 mode		REL3 mode		REL4 mode		0	0

Mode	Function	
b 0 0	UNCHANGED	no effect on the status of the signal emitter
b 0 1	ON	signal emitter on
b 1 0	OFF	signal emitter off
b 1 1	FLASH	signal emitter alternating on

The assignment of the configuration block CFG2 is shown above. The parameter IDLE-MODE occupies two bytes and contains sub parameters for four relays and one digital output. Each output can be configured for one of four states according to the table. As the IDLE-MODE field is not greyed out, the modification can be made in the RAM of the reader.

The following steps are now necessary for the modification of REL1 mode inside IDLE-MODE:

```
// the example shows the reading, modification and rewriting of one block of the reader configuration
// reader is an object of the reader class FedmlscReader

byte CfgAdr = 2;           // Address of the configuration block
bool EEPROM = false;      // Configuration data from/in RAM of the reader
uint IdleModeRel1         // Parameter IDLE-MODE

// Defaults for the next SendProtocol
reader.SetData(OBID.ReaderCommand._0x80.Req.CFG_ADDRESS, (byte)0x00); // reset everything
reader.SetData(OBID.ReaderCommand._0x80.Req.CfgAddr.ADDRESS, CfgAdr);   // set address
reader.SetData(OBID.ReaderCommand._0x80.Req.CfgAddr.LOCATION, EEPROM);  // set memory location on RAM

// read configuration data
reader.SendProtocol(0x80);

IdleModeRel1 = 3;          // REL1 alternating on (Note: set frequency in Parameter IDLE-FLASH)

reader.SetConfigPara(OBID.ReaderConfig.DigitalIO.Relay.No1.IdleMode, IdleMode, false); // change value in RAM
```

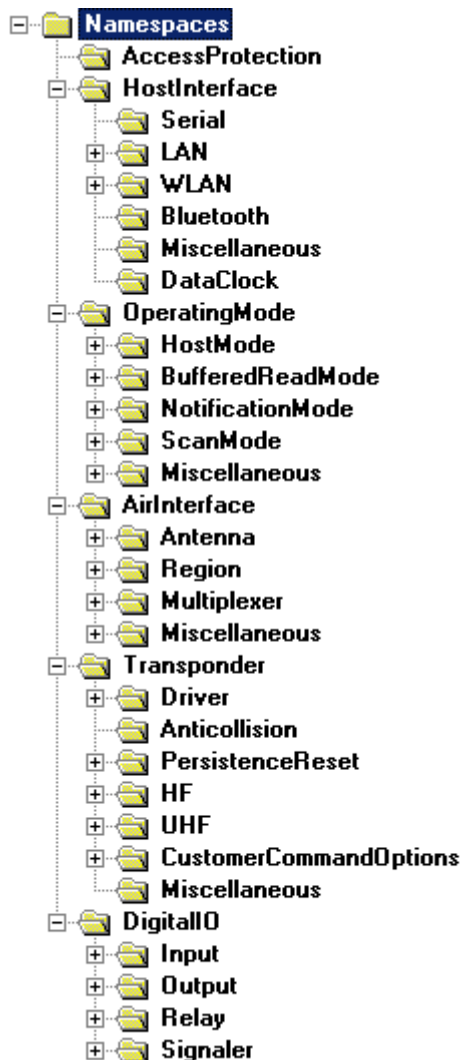
```
// Defaults for the next SendProtocol
```

```
reader.SetData(OBID.ReaderCommand._0x81.Req.CFG_ADDRESS, (byte)0x00); // reset everything
reader.SetData(OBID.ReaderCommand._0x81.Req.CfgAddr.ADDRESS, CfgAdr); // set address
reader.SetData(OBID.ReaderCommand._0x81.Req.CfgAddr.LOCATION, EEPROM); // set memory location on EEPROM
```

```
// rewrite configuration data
```

```
reader.SendProtocol(0x81);
```

The methods *GetConfigPara* and *SetConfigPara* receive a string with parameter name from the namespace **OBID.ReaderConfig**. This main namespace contains further namespaces in tree order and collects all parameter names of all OBID i-scan® and OBID®classic-pro reader in a unique manner. The picture below shows the main namespaces.



The advantage of this schematic is the support by the intellisense functionality of modern IDEs which speeds-up the search for the proper parameter name.

5.6.5. Serializing

The integrated method *Serialize* allows saving the reader configuration from the data containers to a file or loading the reader configuration from a file into data containers.

The standardizing of XML (Extensible Markup Language) has enabled an accepted description language for documents, which can be used independently of the computer language and operating systems. It therefore makes sense to use this language for defining the structure of a reader configuration file. Following is the content of an XML file that was created using the program ISOStart:

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<OBID>
  <file-header>
    <document-type>Reader Configuration File</document-type>
    <document-version>1.0</document-version>
    <reader-family>ISC</reader-family>
    <reader-name>ID ISC.MR100</reader-name>
    <reader-type>74</reader-type>
    <host-address>192.168.3.3</host-address>
    <port-number>10001</port-number>
    <communication-mode>TCP</communication-mode>
    <program-name>ID ISOStart</program-name>
    <program-version>05.03.03</program-version>
    <fedm-version>01.08</fedm-version>
    <date>07/18/03</date>
    <time>11:13:28</time>
  </file-header>
  <data-array name="Reader EEPROM-Parameter" blocks="16" size="16">
    <CFG0 b0="00" b1="00" b2="00" b3="00" b4="00" b5="00" b6="00" b7="00" b8="00" b9="00" b10="00"
      b11="00" b12="00" b13="00" b14="00" b15="00"/>
    <CFG1 b0="00" b1="00" b2="08" b3="01" b4="00" b5="00" b6="00" b7="0A" b8="00" b9="00" b10="00"
      b11="00" b12="00" b13="00" b14="00" b15="00"/>
    <CFG2 b0="00" b1="20" b2="00" b3="25" b4="00" b5="04" b6="00" b7="2F" b8="0A" b9="64" b10="00"
      b11="00" b12="00" b13="00" b14="00" b15="00"/>
    <CFG3 b0="00" b1="39" b2="00" b3="07" b4="00" b5="00" b6="06" b7="00" b8="00" b9="00" b10="00"
      b11="00" b12="00" b13="00" b14="00" b15="00"/>
    <CFG4 b0="00" b1="00" b2="00" b3="00" b4="09" b5="00" b6="00" b7="00" b8="00" b9="00" b10="00"
      b11="00" b12="00" b13="00" b14="00" b15="00"/>
    <CFG5 b0="00" b1="00" b2="00" b3="00" b4="00" b5="00" b6="00" b7="00" b8="00" b9="00" b10="00"
      b11="04" b12="00" b13="00" b14="00" b15="00"/>
    <CFG6 b0="00" b1="00" b2="00" b3="01" b4="00" b5="00" b6="00" b7="0A" b8="00" b9="00" b10="00"
      b11="05" b12="04" b13="00" b14="00" b15="00"/>
    <CFG7 b0="02" b1="20" b2="2C" b3="01" b4="0D" b5="00" b6="00" b7="00" b8="00" b9="00" b10="00"
      b11="00" b12="00" b13="00" b14="00" b15="00"/>
    <CFG8 b0="00" b1="00" b2="00" b3="00" b4="00" b5="00" b6="00" b7="00" b8="00" b9="00" b10="00"
      b11="00" b12="00" b13="00" b14="00" b15="00"/>
    <CFG9 b0="00" b1="00" b2="00" b3="00" b4="00" b5="00" b6="00" b7="00" b8="00" b9="00" b10="00"
      b11="00" b12="00" b13="00" b14="00" b15="00"/>
    <CFG10 b0="00" b1="00" b2="00" b3="00" b4="00" b5="00" b6="00" b7="00" b8="00" b9="00"
      b10="00" b11="00" b12="00" b13="00" b14="00" b15="00"/>
    <CFG11 b0="00" b1="00" b2="00" b3="00" b4="00" b5="00" b6="00" b7="00" b8="00" b9="00"
      b10="00" b11="00" b12="00" b13="00" b14="00" b15="00"/>
    <CFG12 b0="00" b1="00" b2="00" b3="00" b4="00" b5="00" b6="00" b7="00" b8="00" b9="00"
      b10="00" b11="00" b12="00" b13="00" b14="00" b15="00"/>
    <CFG13 b0="00" b1="00" b2="00" b3="00" b4="00" b5="00" b6="00" b7="00" b8="00" b9="00"
      b10="00" b11="00" b12="00" b13="00" b14="00" b15="00"/>
    <CFG14 b0="00" b1="00" b2="00" b3="00" b4="00" b5="00" b6="00" b7="00" b8="00" b9="00"
      b10="00" b11="00" b12="00" b13="00" b14="00" b15="00"/>
    <CFG15 b0="00" b1="00" b2="00" b3="00" b4="00" b5="00" b6="00" b7="00" b8="00" b9="00"
      b10="00" b11="00" b12="00" b13="00" b14="00" b15="00"/>
  </data-array>
  <data-array name="Reader RAM-Parameter" blocks="16" size="16">
    <CFG0 b0="00" b1="00" b2="00" b3="00" b4="00" b5="00" b6="00" b7="00" b8="00" b9="00" b10="00"
      b11="00" b12="00" b13="00" b14="00" b15="00"/>
    <CFG1 b0="00" b1="00" b2="00" b3="00" b4="00" b5="00" b6="00" b7="00" b8="00" b9="00" b10="00"
      b11="00" b12="00" b13="00" b14="00" b15="00"/>
    <CFG2 b0="00" b1="00" b2="00" b3="00" b4="00" b5="00" b6="00" b7="00" b8="00" b9="00" b10="00"
      b11="00" b12="00" b13="00" b14="00" b15="00"/>
    <CFG3 b0="00" b1="00" b2="00" b3="00" b4="00" b5="00" b6="00" b7="00" b8="00" b9="00" b10="00"
      b11="00" b12="00" b13="00" b14="00" b15="00"/>
    <CFG4 b0="00" b1="00" b2="00" b3="00" b4="00" b5="00" b6="00" b7="00" b8="00" b9="00" b10="00"
      b11="00" b12="00" b13="00" b14="00" b15="00"/>
  </data-array>
</OBID>
```

```

b11="00" b12="00" b13="00" b14="00" b15="00"/>
<CFG5 b0="00" b1="00" b2="00" b3="00" b4="00" b5="00" b6="00" b7="00" b8="00" b9="00" b10="00"
b11="00" b12="00" b13="00" b14="00" b15="00"/>
<CFG6 b0="00" b1="00" b2="00" b3="00" b4="00" b5="00" b6="00" b7="00" b8="00" b9="00" b10="00"
b11="00" b12="00" b13="00" b14="00" b15="00"/>
<CFG7 b0="00" b1="00" b2="00" b3="00" b4="00" b5="00" b6="00" b7="00" b8="00" b9="00" b10="00"
b11="00" b12="00" b13="00" b14="00" b15="00"/>
<CFG8 b0="00" b1="00" b2="00" b3="00" b4="00" b5="00" b6="00" b7="00" b8="00" b9="00" b10="00"
b11="00" b12="00" b13="00" b14="00" b15="00"/>
<CFG9 b0="00" b1="00" b2="00" b3="00" b4="00" b5="00" b6="00" b7="00" b8="00" b9="00" b10="00"
b11="00" b12="00" b13="00" b14="00" b15="00"/>
<CFG10 b0="00" b1="00" b2="00" b3="00" b4="00" b5="00" b6="00" b7="00" b8="00" b9="00"
b10="00" b11="00" b12="00" b13="00" b14="00" b15="00"/>
<CFG11 b0="00" b1="00" b2="00" b3="00" b4="00" b5="00" b6="00" b7="00" b8="00" b9="00"
b10="00" b11="00" b12="00" b13="00" b14="00" b15="00"/>
<CFG12 b0="00" b1="00" b2="00" b3="00" b4="00" b5="00" b6="00" b7="00" b8="00" b9="00"
b10="00" b11="00" b12="00" b13="00" b14="00" b15="00"/>
<CFG13 b0="00" b1="00" b2="00" b3="00" b4="00" b5="00" b6="00" b7="00" b8="00" b9="00"
b10="00" b11="00" b12="00" b13="00" b14="00" b15="00"/>
<CFG14 b0="00" b1="00" b2="00" b3="00" b4="00" b5="00" b6="00" b7="00" b8="00" b9="00"
b10="00" b11="00" b12="00" b13="00" b14="00" b15="00"/>
<CFG15 b0="00" b1="00" b2="00" b3="00" b4="00" b5="00" b6="00" b7="00" b8="00" b9="00"
b10="00" b11="00" b12="00" b13="00" b14="00" b15="00"/>
</data-array>
</OBID>

```

Along with some header data, the tags `<data-array name="Reader EEPROM-Parameter" blocks="16" size="16">` and `<data-array name="Reader RAM-Parameter" blocks="16" size="16">` contain the reader parameters as hex values.

The *Serialize* method can be used to create this file or read the reader configuration of such a file and place it in the internal memory *EEData* or *RAMData*. The prerequisite for generating the configuration file is that the entire reader configuration has first been read using *SendProtocol*.

To create a reader configuration file, use the call:

```
Serialize(false, "c:\\tmp\\myreader.xml")
```

and to read the data from a reader configuration file, use the call:

```
Serialize(true, "c:\\tmp\\myreader.xml")
```

5.7. Tables

OBID *i-scan*® and OBID® *classic-pro* readers support protocols that can transport data for multiple transponders (ISO-Host Mode, Buffered Read Mode, Notification Mode) which make saving in the containers impossible. Ideally these data are structure in a table. The reader class **FedmlscReader** contains the tables ISOTable and BRMTable for these transponder data. Access to the table data is possible using the methods *GetTableData*, *SetTableData* and *FindTableIndex*. The methods *GetTableSize*, *SetTableSize*, *GetTableLength* and *ResetTable* are for table administration. In addition, the method *VerifyTableData* can be used to perform a comparison of the sent with the received transponder data (ISO-Host Mode only).

Access to table data using the methods *SetTableData* and *GetTableData* is also accomplished using the methods *SetData* and *GetData* for data containers. But they do not represent a string and therefore do not provide location coding. Instead, unambiguous identification of a table value is possible with the table index (idx) and the constants for the table type (tableID) and the table variable (dataID).

Example:

```
int GetTableData(int index, int tableID, int dataID )
```

All constants for the table type and for the table variables are contained in the interface **FedmlscReaderConst**.

Alternately, tables can be output as table objects of type **FedmlsoTableItem[]** or **FedmBrmTableItem[]** using the method *getTable*. Use of the method interface of the table classes is analogous. Using the method *settable* you can also transfer a table created and filled in the application to the reader class and then write these data to the transponder.

The methods *getTableItem* and *setTableItem* permit the exchange of individual table elements.

Important note: A new reader object has unsized tables. You must therefore immediately set the size of your table using the method *setTableSize* (see [5.1.1. Initializing](#)).

5.8. Communication with Transponders in the Host-Mode

Programmers have two alternatives in the Reader class `FedmlscReader` for the communication with Transponders:

1. Table oriented API (s. [8.2. Table oriented commands](#))
2. TagHandler API, based on specialized Transponder classes

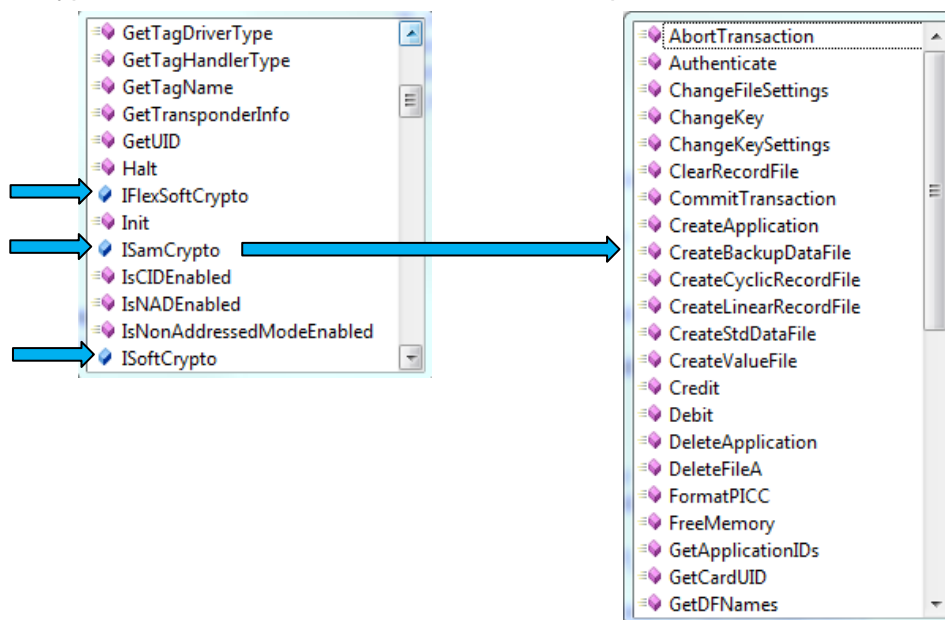
It is recommended to use the 2nd API for new projects. TagHandler classes are specific to Transponder standards like ISO 14443, ISO 15693 and EPC Class 1 Gen 2 or customized for manufacturer specific extended API. Each standard and chip type is implemented as a class and all classes together build a hierarchical system of derived classes. Base class is `FedmlscTagHandler`.

Precondition for the use of TagHandler classes is 1st the use of the methods `TagInventory` and `TagSelect` from the Reader class `FedmlscReader` and 2nd the identifiability of the Transponder standard and/or chip type for the accurate creation of TagHandler classes. Unsupported chip types are assigned automatically to the base class `FedmlscTagHandler`.

TagHandler objects are managed by the table `FedmlscTableItem`. Thus, they use internally the table oriented API.

The method interface of each TagHandler class is made up of the command list of Transponder standards or chip types. Consequently, the programmer has to work with the documentation of a standard or with the Transponder manual from the manufacturer to understand the meaning of the method interfaces.

The following picture demonstrates with the example of the ISO 14443-A Transponder MIFARE DESFire the method interface (left) of the TagHandler class `FedmlscTagHandler_ISO14443_4_MIFARE_DESFire` and, after selection of the internal interface – here: `ISamCrypto` – the real method interface of the Transponder.



H40301-e-ID-B.docx



6. Error handling

6.1. Return value

Many methods in the class library perform internal error diagnostics and in case of an error return a negative value. The error codes for the Java class library ID OBIDISC4J have been directly taken from the native implementations. They are organized into ranges so that they do not overlap. The following ranges are reserved for the C++ class library ID FEDM and the native OBID®-function libraries:

Library	Value range for error codes	Reference
ID FEDM	-101 ... -999	10.1.List of error codes
ID FECOM	-1000...-1099	H80592-xx-ID-B
ID FEUSB	-1100...-1199	H00501-xx-ID-B
ID FETCP	-1200...-1299	H30802-xx-ID-B
ID FEISC	-4000...-4099	H9391-xx-ID-B
ID FEFU	-4100...-4199	H30801-xx-ID-B
IF FETCL	-4200...-4299	H50401-xx-ID-B

The method *GetErrorText* of the reader class can be used to get an error text for the error code. The error code can also come from the area of a native OBID®-function library.

The last error code is saved internally and can be retrieved using the method *GetLastError*.

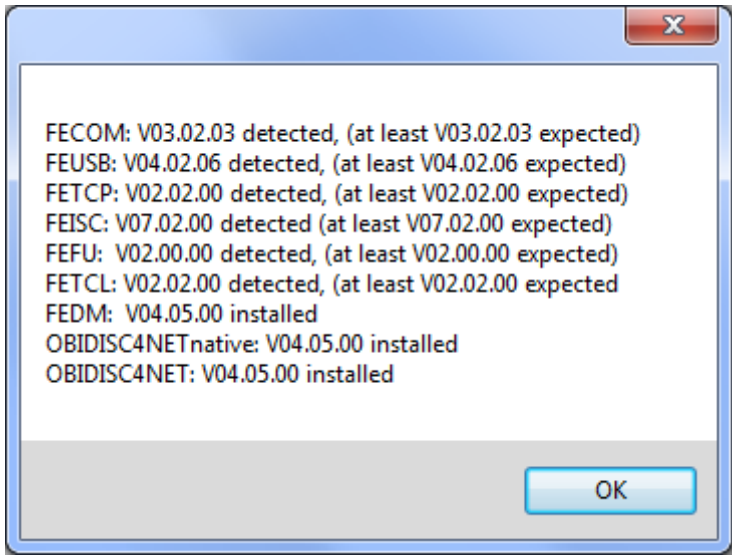
6.2. Exceptions

Exceptions are generated in exceptional situations in the wrapper class for .NET and during communication. This manual explains for each method whether and which exceptions are generated.

7. Library Reference

7.1. FedmlscReader

7.1.1. GetDependentLibVersions

Function	Method returns a string with version information.
Syntax	string GetDependentLibVersions()
Description	This method returns a string with version information of all dependent libraries. This can be useful for log files or for information display.
Exceptions	-
Example	<pre>using OBID; ... string Info = reader.GetDependentLibVersions(); Console.WriteLine(Info);</pre>  <p>The screenshot shows a standard Windows message box with a blue title bar and a red 'X' button in the top right corner. The text inside the box lists the following information:</p> <ul style="list-style-type: none">FECOM: V03.02.03 detected, (at least V03.02.03 expected)FEUSB: V04.02.06 detected, (at least V04.02.06 expected)FETCP: V02.02.00 detected, (at least V02.02.00 expected)FEISC: V07.02.00 detected, (at least V07.02.00 expected)FEFU: V02.00.00 detected, (at least V02.00.00 expected)FETCL: V02.02.00 detected, (at least V02.02.00 expected)FEDM: V04.05.00 installedOBIDISC4NETnative: V04.05.00 installedOBIDISC4NET: V04.05.00 installed <p>An 'OK' button is located at the bottom right of the message box.</p>

7.1.2. FedmlscReader

Function	Generates a new instance of the class FedmlscReader (Constructor).
Syntax	FedmlscReader()
Description	A Reader object is generated. Only a Reader object can be used to run the protocol functions.
Return value	If a Reader object was able to be created without error, the new instance of the class FedmlscReader is returned.
Exceptions	In case of error the method throws the exception FedmException .
Example	<pre>using OBID; ... try { reader = new FedmlscReader(); // generate new instance } catch (FedmException e) { Console.WriteLine("Exception when generating a new Reader object: " + e.Message); } private FedmlscReader reader;</pre>

7.1.3. ConnectCOMM

Function	Opens a serial port and and optionally detects a Reader
Syntax	void ConnectCOMM(int portNumber, bool withDetect) void ConnectCOMM(int portNumber, bool withDetect uint authenticType, string authenticKey)
Description	<p>A serial port is opened.</p> <p>Optionally, a connected Reader is detected with the internal execution of <i>FindBaudrate</i>. If the detection was successful the method <i>ReadReaderInfo</i> is called internally to query important reader information and to set the reader type.</p> <p>In case of an error, the serial port is closed at once.</p> <p>The parameter <i>portNumber</i> must be between 1 and 256.</p> <p>This method can be used with any Reader object. To change the port, the opened port must first be closed using the method <i>Disconnect</i>.</p> <p>The second method opens a connection and establishes a secured transmission.</p> <p><i>authenticType</i>: 0 – AES-128 1 – AES-192 2 – AES-256</p> <p><i>authenticKey</i> contains the password as a string with hexadecimal chars (0..9, A..F). The length of the password depends on the key type: AES-128 - 32 chars AES-192 - 48 chars AES-256 - 64 chars</p>
Exceptions	In case of error the method throws one of the exceptions FedmException , FePortDriverException , FeReaderDriverException .
Example	<pre>using OBID; ... reader.ConnectCOMM(1, false); // Open COM1 reader.FindBaudrate(); // Baudrate and Frame will be detected reader.ReadReaderInfo(); // query reader information and set reader type // or alternatively with detect of a connected reader reader.ConnectCOMM(1, true); ... private FedmIscReader reader;</pre>

7.1.4. ConnectTCP

Function	Opens the connection to a server via TCP/IP.
Syntax	void ConnectTCP(string host, int portNumber) void ConnectTCP(string host, int portNumber, uint authenticType, string authenticKey)
Description	<p>These methods open a connection to a TCP/IP server in the network. The port number should lie between 1024 and 65535 in order to avoid conflicts with the „well known ports“.</p> <p>This method can be used only once for each Reader object. To change the connection, the opened connection must first be closed using the method Disconnect.</p> <p>The second method opens a connection and establishes a secured transmission.</p> <p><i>authenticType</i>: 0 – AES-128 1 – AES-192 2 – AES-256</p> <p><i>authenticKey</i> contains the password as a string with hexadecimal chars (0..9, A..F). The length of the password depends on the key type: AES-128 - 32 chars AES-192 - 48 chars AES-256 - 64 chars</p>
Exceptions	In case of error the method throws one of the exceptions FedmException , FePortDriverException , FeReaderDriverException .
Example	<pre>using OBID; ... reader.ConnectTCP(„192.168.1.127“, 10001); // Connection to server at // address „192.168.1.127“ ... private FedmIscReader reader;</pre>

7.1.5. ConnectUSB

Function	Opens the connection to a Reader via USB.
Syntax	void ConnectUSB(long deviceId) void ConnectUSB(long deviceId, uint authenticType, string authenticKey)
Description	<p>The method opens the connection to a Reader via USB.</p> <p>If a 0 is given for the parameter <i>deviceId</i>, then the first scanned USB Reader is used. In this case only one USB Reader should be connected to the PC.</p> <p>The second method opens a connection and establishes a secured transmission.</p> <p><i>authenticType</i>: 0 – AES-128 1 – AES-192 2 – AES-256</p> <p><i>authenticKey</i> contains the password as a string with hexadecimal chars (0..9, A..F). The length of the password depends on the key type: AES-128 - 32 chars AES-192 - 48 chars AES-256 - 64 chars</p> <p>If multiple USB Readers are connected and need to be processed at the same time, the help class FeUsb can be used. This allows the USB bus to be scanned for Readers.</p> <p>To change the connection, the opened connection must first be closed using the method Disconnect.</p>
Exceptions	In case of error the method throws one of the exceptions FedmException , FePortDriverException , FeReaderDriverException .
Example	<pre>using OBID; ... reader.ConnectUSB(0); // Open connection to USB Reader ... private FedmIsCReader reader;</pre>

7.1.6. Disconnect

Function	This method closes the connection between Reader and PC and frees up reserved memory.
Syntax	int Disconnect()
Description	<p>This method closes the connection between Reader and PC and frees up reserved memory.</p> <p>If closing a TCP/IP connection, the return value reflects the last status. If the status is TIME_WAIT, then a 0 is returned to indicate a successful disconnection.</p> <p>See also: 10.4. TCP Status</p>
Exceptions	In case of error the method throws one of the exceptions FedmException , FePortDriverException
Example	<pre>using OBID; ... reader.ConnectUSB(0); // Open connection to USB Reader if (reader.Connected) { back = reader.Disconnect(); // Close connection to reader } ... private FedmIsCReader reader;</pre>

7.1.7. GetTcpConnectionStatus

Function	Returns the actual state of a TCP/IP connection
Syntax	int GetTcpConnectionStatus() int GetTcpConnectionState(string HostAdr, uint PortNumber)
Description	<p>Detects with a Kernel function the status of a TCP/IP connection.</p> <p>The 1st method can be used after a call of ConnectTCP() and before Disconnect().</p> <p>The 2nd method can be used after a call of Disconnect(), when the TCP-Status is not TIME_WAIT to observe the connection, until TIME_WAIT is reached.</p> <p>It is not possible to get the status with continuous polling to detect a broken connection caused by loss of power or lost network cable. This method can help to find reasons <u>after</u> a communication error.</p> <p>See also: 10.4. TCP Status</p>
Exceptions	In case of error the method throws one of the exceptions FedmException , FePortDriverException
Example	

7.1.8. SetPortPara

Function	Sets new parameters for the serial port.					
Syntax	void SetPortPara (string parameter, string val)					
Description	Sets new parameters for the serial port. The following parameters are allowed:					
	Parameter	Value range	Default	Unit	Port	Description
	Baud	300...115200	9600	bit/s	<input checked="" type="checkbox"/> Seriell <input type="checkbox"/> USB <input type="checkbox"/> TCP/IP	Port baud rate
	Frame	7N1, 7E1, 7O1, 7N2, 7E2, 7O2, 8N1, 8E1, 8O1	8E1		<input checked="" type="checkbox"/> Seriell <input type="checkbox"/> USB <input type="checkbox"/> TCP/IP	Character frame (data bits, parity, stop bits)
	Timeout	0...99999	600	ms	<input checked="" type="checkbox"/> Seriell <input checked="" type="checkbox"/> USB <input checked="" type="checkbox"/> TCP/IP	Maximum wait time for receive protocol
	TxTimeControl	0, 1	1	-	<input checked="" type="checkbox"/> Seriell <input type="checkbox"/> USB <input type="checkbox"/> TCP/IP	If set (1), then the output of the next send protocol is delayed internally until at least TxDelayTime (ms) after the last receive protocol has elapsed. If not set (0), then the send protocol is always output as soon as possible.
	TxDelayTime	0...999	5	ms	<input checked="" type="checkbox"/> Seriell <input type="checkbox"/> USB <input type="checkbox"/> TCP/IP	Minimum time between the 1 st receive and next send protocol. Is only applied if TxTimeControl=1
	CharTimeoutMp y	1...99	1	-	<input checked="" type="checkbox"/> Seriell <input type="checkbox"/> USB <input type="checkbox"/> TCP/IP	The character timeout for the serial port is calculated internally. The character timeout determines after how much time after receipt of the last character the receive process is ended. With some PCs this can result in repeated protocol length errors because the wait time is too short. In such cases this parameter can be used to multiply the wait time.
	SetDTR	0, 1	0	-	<input checked="" type="checkbox"/> Seriell <input type="checkbox"/> USB <input type="checkbox"/> TCP/IP	Set of the DTR control line
	SetRTS	0, 1	0	-	<input checked="" type="checkbox"/> Seriell <input type="checkbox"/> USB <input type="checkbox"/> TCP/IP	Set of the RTS control line
Exceptions	In case of error the method throws one of the exceptions FedmException , FePortDriverException , FeReaderDriverException .					
Example	using OBID; ...					

	<pre>reader.ConnectCOMM(1); // Open connection to COM Reader reader.setPortPara(„Baud“, „9600“); // Change baud rate ... private FedmIscReader reader;</pre>
--	--------------------------------------------------------------------------------------------------------------------------------------------------------------

7.1.9. GetPortPara

Function	Queries the parameters for the serial port.
Syntax	string GetPortPara(string parameter)
Description	Queries the parameters for the serial port. For additional information see section SetPortPara
Exceptions	In case of error the method throws one of the exceptions FedmException , FePortDriverException , FeReaderDriverException .
Example	<pre>using OBID; ... reader.ConnectCOMM(1); // Open connection to COM Reader string baud = reader.GetPortPara(„Baud“); // Get baud rate ... private FedmIsCReader reader;</pre>

7.1.10. SetBusAddress

Function	Sets the bus address in the Reader object.
Syntax	void SetBusAddress(byte busAddress)
Description	The method sets the bus address in the Reader object for communication. The default address is 255 and can be used if only one Reader is connected to the serial port.
Exceptions	None
Example	<pre>using OBID; ... reader.ConnectCOMM(1); // Open connection to COM Reader reader.SetBusAddress(1); // Set bus address to 1 ... private FedmIsCReader reader;</pre>

7.1.11. GetBusAddress

Function	Gets the bus address from a Reader object.
Syntax	byte GetBusAddress()
Description	The method gets the bus address from a Reader object.
Return Value	The method returns the bus address
Exceptions	None
Example	<pre>Using OBID; ... reader.ConnectCOMM(1); // Open connection to COM Reader byte busAddress = reader.GetBusAddress(); // Query the bus address Console.Write(„The set bus address is: „); Console.WriteLine(busAddress); ... private FedmIscReader reader;</pre>

7.1.12. GetFamilyCode

Function	Returns a short string with the family code.
Syntax	string GetFamilyCode()
Description	Returns a short string with the family code.
Return Value	„ISC“
Exceptions	None
Example	<pre>Using OBID; ... reader.ConnectCOMM(1); // Open connection to COM Reader reader.sendProtocol(0x65); // GetSoftVersion string familyCode = reader.GetFamilyCode(); // Query the Reader family Console.WriteLine(„The Reader is a „ + familyCode + „ – Reader“); ... private FedmIscReader reader;</pre>

7.1.13. GetReaderName

Function	Returns a short string with the Reader name.
Syntax	String GetReaderName()
Description	Returns a short string with the Reader name. The software version must first have been read.
Return Value	A short string with the Reader name.
Exceptions	None
Example	<pre>using OBID; ... reader.ConnectCOMM(1); // Open connection to COM Reader reader.sendProtocol(0x65); // GetSoftVersion string readerName = reader.GetReaderName(); // Query the Reader name Console.WriteLine(„The Reader has the sales name: „ + readerName); ... private FedmIscReader reader;</pre>

7.1.14. GetTransponderName

Function	Returns a short string with the transponder name.
Syntax	String GetTransponderName(byte type)
Description	Returns a short string with the transponder name. The parameter type contains the transponder type according the system manual of the reader.
Return Value	A short string with the transponder name.
Exceptions	None
Example	<pre>using OBID; ... string trpName = reader.GetTransponderName(FedmIscReaderConst.TR_TYPE_EPC); Console.WriteLine(„The transponder has the name: „ + trpName); ... private FedmIscReader reader;</pre>

7.1.15. GetReaderType

Function	Returns the Reader number. The software version must first have been read.
Syntax	int GetReaderType()
Description	Returns the Reader type number. The software version must first have been read.
Return Value	An integer with the Reader type.
Exceptions	None
Example	<pre>using OBID; ... reader.ConnectCOMM(1); // Open connection to COM Reader reader.sendProtocol(0x65); // GetSoftVersion int readerType = reader.GetReaderType(); // Query the Reader type Console.Write(„The Reader is of type: „); Console.WriteLine(readerType); ... private FedmIsCReader reader;</pre>

7.1.16. SetReaderType

Function	Sets a new Reader type number.
Syntax	void SetReaderType(int readerType)
Description	<p>This method sets the Reader type and part number of the Reader. The Reader type must have been defined in the system manual.</p> <p>This method is automatically opened after invoking <i>SendProtocol(0x65)</i>, with which the software and Reader version is read.</p> <p>All Reader type constants are listed in the structure FedmIsCReaderConst.</p>
Exceptions	FedmException , if a <i>readerType</i> is unknown
Example	<pre>using OBID; ... reader.ConnectCOMM(1); // Open connection to COM Reader int readerType = reader.SetReaderType(41); // The Reader is a ID ISC.LR200 ... private FedmIsCReader reader;</pre>

7.1.17. SendProtocol

Function	The central communications method.
Syntax	(1) Int SendProtocol(byte cmd) (2) int SendProtocol(byte cmd, string RequestData, out string ResponseData)
Description	<p>Protocol traffic is carried out using the method <i>SendProtocol</i> (1). This will pass only the control byte for the selected protocol. All data needed for the protocol transfer are taken from the data containers and data tables. Therefore you must ensure that all required parameters have been previously set.</p> <p>A second overloaded implementation (2) can be used, if the library has no support for it (e.g. some [0xB1] ISO15693 Customer and Proprietary Commads). The parameter <i>cmd</i> contains the command byte and the parameter <i>RequestData</i> a string with all data bytes, following after the command byte. The parameter <i>ResponseData</i> contains the responded data.</p> <p>The method can be used with all port drivers.</p>
Return Value	An integer with the error code (< 0) or OK (0) or Statusbyte of Reader response (>0)
Exceptions	In case of error the method throws one of the exceptions FedmException , FePortDriverException , FeReaderDriverException .
Example	For a detailed description of how to use this method, see 8.Examples for using the function SendProtocol .

7.1.18. SendTransparent

Function	Communication function for rare cases
Syntax	int SendTransparent(string SndProtocol, bool CalcChecksum, out string RecProtocol)
Description	<p>The function <i>SendTransparent</i> allows the construction and exchange of protocols with protocol frame under control of the application. Deep knowledge about protocol frames is necessary. The parameter <i>CalcChecksum</i> enables the option to calculate and add of the crc checksum internally.</p> <p>The method can be used with all port drivers.</p>
Return Value	An integer with the error code (< 0) or length of responded protocol(≥ 0).
Exceptions	In case of error the method throws one of the exceptions FedmException , FePortDriverException , FeReaderDriverException .
Example	

7.1.19.TagInventory

Function	Communication method for Inventory
Syntax	Dictionary<string, FedmIscTagHandler> TagInventory(bool bAll, byte ucMode, byte ucAntennas)
Description	<p>An Inventory command is executet on the RF interface. For each detected Transponder a TagHandler class will be created and returnd in a Dictionary. The key of each Dictionary entry is the UID of a Transponder.</p> <p>The parameters <i>bAll</i>, <i>ucMode</i>and <i>ucAntennas</i> controls the Inventory and are identical with the Inventory parameters in the Reader's System Manual. In the standard case thefollowing values are properly:<i>bAll</i>(true), <i>ucMode</i>(0x00)and <i>ucAntennas</i>(1).</p>
Return Value	Dictionary with all detected Transpondersor an empty Dictionary, if no Transponder is detected.
Exceptions	In case of error the method throws one of the exceptions FedmException , FePortDriverException , FeReaderDriverException .
Example	<pre> using OBID; using OBID.TagHandler; Dictionary<string, FedmIscTagHandler> TagList; Dictionary<string, FedmIscTagHandler>.ValueCollection listTagHandler; FedmIscTagHandler TagHandler = null; FedmIscTagHandler_ISO14443_4_MIFARE_DESFire DESFire = null; // Inventory command TagList = Reader.TagInventory(true, 0x00, 1)); if (TagList.Count> 0) { listTagHandler = TagList.Values; foreach (FedmIscTagHandler tagHandler in listTagHandler) { if (tagHandler != null) { TagHandler = tagHandler; break;// we break as we want to have the first item } } // Try to select tranponder as Mifare DESFire TagHandler = Reader.TagSelect(TagHandler, 9); if (TagHandler isFedmIscTagHandler_ISO14443_4_MIFARE_DESFire) { DESFire = (FedmIscTagHandler_ISO14443_4_MIFARE_DESFire)TagHandler; // do anything with this DESFire-TagHandler } } </pre>

7.1.20.TagSelect

Function	Communication method for Select-Command for one Transponder
Syntax	FedmlscTagHandler TagSelect(FedmTagHandler tagHandler, uint tagDriver)
Description	<p>A Select command is executed on the RF interface. If the Select was successful, the TagHandler of the same type is updated or, mostly in cases for ISO 14443 compliant Transponders, a new TagHandler of a more specialized type is created and returned.</p> <p>The optional 2nd parameter <i>tagDriver</i> defines the Transponder driver to be used by the Reader. Detailed information to this parameter can be found in the Reader's System Manual. With a 0 for <i>tagDriver</i>, the parameter is deactivated.</p> <p>The method can be used with all port drivers.</p>
Return Value	TagHandler of selected Transponder or null in error case.
Exceptions	In case of error the method throws one of the exceptions FedmException , FePortDriverException , FeReaderDriverException .
Example	s. TagInventory

7.1.21. SendTclApu. SendTclPing, SendTclDeselect

Function	Communication function for APDUs
Syntax	int SendTclApu(FedmCprApu apdu) int SendTclPing(FedmCprApu apdu) int SendTclDeselect(FedmCprApu apdu)
Description	<p>APDU exchange with an ISO14443-4 transponder. The execution is asynchronous and the application will be notified by the call of OnNewApuResponse.</p> <p>The methods SendTclPing and SendTclDeselect are executed synchronous.</p> <p>These methods can be used with all port drivers.</p> <p>When an APDU process is finished by the Transponder with an ISO Error, then the Reader returns the status 0x96 (ISO Error) and adds the ISO 14443 error code to the protocol. In this case, the additional ISO error code can be requested inside the OnNewApuResponse (see example below).</p> <p>In case of an ISO 14443 error together with Ping or Deselect, the ISO 14443 error code is part of the FedmCprApu runtime object.</p> <p>Please refer to the Reader's system manual to get detailed information about the error codes.</p>
Return Value	An integer with the error code (< 0) or OK (0). For SendTclPing and SendTclDeselect a value >0 reflects the Statusbyte the of Reader response.
Exceptions	In case of error the method throws one of the exceptions FedmException , FePortDriverException , FeReaderDriverException .
Example	<pre> using OBID; // reader is of type FedmIscReader // one APDU object as a member in your application FedmCprApu apdu = new FedmCprApu(this); public void SendApu() { byte[] apduData = new byte[1]; apduData[0] = 0x60; apdu.SetApu(apduData); reader.SendTclApu(apdu); } ... public void CancelApu { Reader.CancelTclApu(); } ... </pre>

```
public void OnNewApuResponse(int iError)
{
    byte[] rspData;
    int isoErrorCode = 0;

    if(iError == 0)
    {
        rspData = apdu.GetLastResponseData();
    }
    else if( iError == 0x96 )
    {
        // ISO 14443 error
        isoErrorCode = apdu.GetIsoErrorCode();
    }
    else if( iError == -4280 )
    {
        // APDU process canceled by user
    }
    else
    {
        // any other error code (<0) or status value (>0) from Reader
    }
}
```

7.1.22. CancelTclApdu

Function	Cancel function for APDU process
Syntax	int CancelTclApdu()
Description	<p>The method CancelTclApdu cancels a running APDU process in the Reader. The method sets a cancel flag and returns immediately. The canceled APDU process is then signaled with the call of OnNewApduResponse with error code -4280 (this is an error code from the library FETCL).</p> <p>Important Note: This function can only cancel APDU processes, when the Reader is connected at USB. Serial and TCP connections are not supported.</p>
Return Value	An integer with the error code (< 0) or OK (0). For SendTclPing and SendTclDeselect a value >0 reflects the Statusbyte the of Reader response.
Exceptions	In case of error the method throws one of the exceptions FedmException , FePortDriverException , FeReaderDriverException .
Example	See 7.1.21. SendTclApdu , SendTclPing , SendTclDeselect

7.1.23. SendCommandQueue

Function	Communication function for [0xBC] Command Queue
Syntax	int SendCommandQueue(FedmCprCommandQueue queue)
Description	<p>The collected commands are sent to the Reader. The execution is asynchronous and the application will be notified by the call of OnNewQueueResponse.</p> <p>The method can be used with all port drivers.</p>
Return Value	An integer with the error code (< 0) or OK (0).
Exceptions	In case of error the method throws one of the exceptions FedmException , FePortDriverException , FeReaderDriverException .
Example	<pre> using OBID; // one command queue object as a member in your application FedmCprCommandQueue queue = new FedmCprCommandQueue(this); SendQueue() { string serialNumber; // get serialNumber after Inventory queue.Clear(); // queue must be first cleared // prepare [0xB0][0x25] Select fedm.SetData(FedmIscReaderID.FEDM_ISC_TMP_B0_MODE, (byte)0x01); // addressed fedm.SetData(FedmIscReaderID.FEDM_ISC_TMP_B0_REQ_UID, serialNumber); fedm.SetData(FedmIscReaderID.FEDM_ISC_TMP_B0_CMD, 0x25); fedm.AddCommand(queue, 0xB0); // prepare [0xB0][0x23] Read Multiple Blocks fedm.SetData(FedmIscReaderID.FEDM_ISC_TMP_B0_MODE, (byte)0x02); // selected fedm.SetData(FedmIscReaderID.FEDM_ISC_TMP_B0_CMD, 0x23); fedm.SetData(FedmIscReaderID.FEDM_ISC_TMP_B0_REQ_DBN, (byte)1); fedm.SetData(FedmIscReaderID.FEDM_ISC_TMP_B0_REQ_DB_ADR, (byte)0); fedm.AddCommandToQueue(queue, 0xB0); fedm.SendCommandQueue(queue); } ... OnNewQueueResponse(int iError) { byte[] rspData; if(iError == 0) rspData = queue.GetLastResponseData(); } </pre>

7.1.24. SendSAMCommand

Function	Communication function for [0xC0] SAM Command
Syntax	(1) int SendSAMCommand(FedmTaskListener l, int iSlot, byte[] SndProtocol, uint uiTimeout) (2) int SendSAMCommand(int iSlot, uint uiTimeout, byte[] SndProtocol, ref byte[] RecProtocol, ref int RecProtocolLen)
Description	A SAM command is sent to the Reader. The execution for method (1) is asynchronous and the application will be notified by the call of OnNewSAMResponse. Method (2) is executed synchronously. The method can be used with all port drivers.
Return Value	An integer with the error code (< 0) or OK(0). For (2) a value >0 reflects the Statusbyte the of Reader response.
Exceptions	In case of error the method throws one of the exceptions FedmException , FePortDriverException , FeReaderDriverException .
Example	

7.1.25. FindBaudrate

Function	Method for finding the baud rate and protocol frame.
Syntax	int FindBaudrate(int readerType)
Description	Detects the baud rate and protocol frame of a Reader and sets the port to the detected parameters. The method can be used only in conjunction with the serial ports.
Return Value	Fedm.OK (0), if a Reader was detected or an error code (< 0)
Exceptions	None
Example	<pre>using OBID; ... reader.ConnectCOMM(1); // Open connection to COM Reader int state = reader.FindBaudrate(); // Find baud rate and protocol frame if (state == 0) { Console.WriteLine(„The Reader was detected with a: „); Console.WriteLine(reader.GetPortPara(„Baud“) + “ baud rate of.”); Console.WriteLine(„The Reader was detected with a: „); Console.WriteLine(reader.GetPortPara(„Frame“) + “ protocol frame of.”); } else { Console.WriteLine(„No reader was detected“); } ... private FedmIscReader reader;</pre>

7.1.26. Serialize

Function	Method for serializing the Reader configuration into or from an XML file.
Syntax	void Serialize(bool read, string fileName)
Description	This method allows loading (if read == true) of a Reader configuration from an XML file into the data container EEPROM and RAM, or writing (if read == false) the contents of the data container into an XML file. The Reader configuration remains unchanged.
Exceptions	In case of error the method throws one of the exceptions FedmException , FePortDriverException , FeReaderDriverException .
Example	

7.1.27. TransferReaderCfgToXmlFile, TransferXmlFileToReaderCfg

Function	Method for serializing the Reader configuration between an XML file and a Reader.
Syntax	int TransferReaderCfgToXmlFile(string fileName) int TransferXmlFileToReaderCfg(string fileName)
Description	The first method reads the complete Reader configuration from a connected Reader and writes this information into an XML file. The Reader configuration remains unchanged. The second method reads the complete Reader configuration from an XML file and writes this information into a connected Reader. The Reader configuration will be modified.
Return Value	An integer with the error code (< 0) or OK (0) or Statusbyte of Reader response (>0)
Exceptions	In case of error the method throws one of the exceptions FedmException , FePortDriverException , FeReaderDriverException .
Example	

7.1.28. ReaderAuthentication

Function	Authentication function
Syntax	int ReaderAuthentication(uint uiAuthentType, string sAuthentKey)
Note	Be careful with this function. You cannot read back the authentication key from the reader.
Description	<p>This function initializes a new secured session. The first session must be established together with the opening of a connection with ConnectTCP.</p> <p><i>authentType</i>: 0 – AES-128 1 – AES-192 2 – AES-256</p> <p><i>authentKey</i> contains the password as a string with hexadecimal chars (0..9, A..F). The length of the password depends on the key type: AES-128 - 32 chars AES-192 - 48 chars AES-256 - 64 chars.</p>
Return Value	Fedm.OK (0) or Statusbyte of Reader response (>0)
Exceptions	In case of error the method throws one of the exceptions FedmException , FePortDriverException , FeReaderDriverException .

7.1.29. ReadReaderInfo

Function	Method reads all important information from reader.
Syntax	int ReadReaderInfo()
Description	Method reads all important information with multiple protocols [0x66] Get Reader Info from reader and saves the values in the helper class FedmlscReaderInfo. It is strongly recommended, to call this method once after the first connection.
Return Value	Instance of FedmlscReaderInfo
Exceptions	In case of error the method throws one of the exceptions FedmException , FePortDriverException , FeReaderDriverException .

7.1.30. ReadCompleteConfiguration

Function	Reads complete reader configuration.
Syntax	int ReadCompleteConfiguration(bool EEPROM)
Description	Method reads from EEPROM (EEPROM=true) respectively RAM (EEPROM=false) all configuration blocks from a reader and saves the values in the internal data container.
Return Value	Fedm.OK (0) or Statusbyte of Reader response (>0)
Exceptions	In case of error the method throws one of the exceptions FedmException , FePortDriverException , FeReaderDriverException .

7.1.31. WriteCompleteConfiguration

Function	Writes complete reader configuration.
Syntax	int WriteCompleteConfiguration(bool EEPROM)
Description	Method writes all configuration blocks from the internal data container to the reader. If the parameter EEPROM is true, the destination inside the reader is the EEPROM and the RAM. Otherwise only the RAM.
Return Value	Fedm.OK (0) or Statusbyte of Reader response (>0)
Exceptions	In case of error the method throws one of the exceptions FedmException , FePortDriverException , FeReaderDriverException .

7.1.32. ResetCompleteConfiguration

Function	Factory reset of complete reader configuration.
Syntax	int ResetCompleteConfiguration(bool EEPROM)
Description	Method executes a configuration reset to factory settings in the internal EEPROM and RAM, if the parameter EEPROM is true or only in the RAM, if EEPROM is false.
Return Value	Fedm.OK (0) or Statusbyte of Reader response (>0)
Exceptions	In case of error the method throws one of the exceptions FedmException , FePortDriverException , FeReaderDriverException .

7.1.33. ApplyConfiguration

Function	Updates the configuration in the reader.
Syntax	int ApplyConfiguration(bool EEPROM)
Description	<p>Method updates the reader configuration in the EEPROM and RAM (EEPROM=true) or only in the RAM (EEPROM=false), after modifying one or multiple parameter values in the data container with <i>SetConfigPara</i>. This method updates only these configuration blocks which are previously modified.</p> <p>The use of this method implies the read of the complete configuration prior the modifications and update of parameter values.</p>
Return Value	Fedm.OK (0) or Statusbyte of Reader response (>0)
Exceptions	In case of error the method throws one of the exceptions FedmException , FePortDriverException , FeReaderDriverException .

7.1.34. GetLastError

Function	Returns the last error code.
Syntax	int GetLastError()
Description	Returns the last error code.
Return Value	Error code (<0)
Exceptions	None
Example	

7.1.35. GetLastErrorStatus

Function	Returns the last status.
Syntax	Byte GetLastErrorStatus()
Description	Returns the status code of the last communication.
Return Value	Status code (>=0)
Exceptions	None
Example	

7.1.36. GetErrorText

Function	Gets an error text.
Syntax	string GetErrorText(int errorCode)
Description	Returns a short error text referenced by the error code.
Return Value	An error text
Exceptions	None
Example	

7.1.37. GetStatusText

Function	Gets a status text
Syntax	string GetStatusText(byte statusCode)
Description	Returns a short status text referenced by the status code.
Return Value	A status text
Exceptions	None
Example	

7.1.38. GetData

Function	Gets a datum from a data container
Syntax	<code>int GetData(string id, out bool data)</code> <code>int GetData(string id, out byte data)</code> <code>int GetData(string id, out byte[] data)</code> <code>int GetData(string id, out uint data)</code> <code>int GetData(string id, out long data)</code> <code>int GetData(string id, out string data)</code> <code>int GetData(int address, out byte[] data, int length, int memID)</code>
Description	<p>Method for reading a value from a data container.</p> <p>The storage location of the datum is determined by the parameter <i>id</i>.</p> <p>The last listed, but rarely used variant of GetData locates the datum by the parameters address and memory identifier (memID).</p> <p>If an error occurs, the method returns an error code (< 0).</p>
Return Value	Fedm.OK or an (< 0)
Exceptions	FedmException
Example	

7.1.39. SetData

Function	Writes a datum into a data container.
Syntax	<code>int SetData(string id, bool data)</code> <code>int SetData(string id, byte data)</code> <code>int SetData(string id, byte[] data)</code> <code>int SetData(string id, uint data)</code> <code>int SetData(string id, long data)</code> <code>int SetData(string id, string data)</code>
Description	<p>Method for setting a value in a data container.</p> <p>The storage location of the datum is determined by the parameter <i>id</i>.</p> <p>If an error occurs, the method returns an error code (< 0).</p>
Return Value	Fedm.OK or an error code (< 0)
Exceptions	FedmException
Example	

7.1.40. GetConfigPara

Function	Gets a configuration parameter value from a data container
Syntax	<code>int GetConfigPara(string id, out bool data, bool EEPROM)</code> <code>int GetConfigPara (string id, out byte data, bool EEPROM)</code> <code>int GetConfigPara (string id, out byte[] data, bool EEPROM)</code> <code>int GetConfigPara (string id, out uint data, bool EEPROM)</code> <code>int GetConfigPara (string id, out long data, bool EEPROM)</code> <code>int GetConfigPara (string id, out string data, bool EEPROM)</code>
Description	<p>Method for reading a configuration parameter value from a data container.</p> <p>The parameter <i>id</i> passes the parameter name from the namespace OBID.ReaderConfig. The last parameter specifies the destination memory EEPROM (true) or RAM (false).</p> <p>If an error occurs, the method returns an error code (< 0).</p>
Return Value	Fedm.OK or an (< 0)
Exceptions	FedmException
Example	

7.1.41. SetConfigPara

Function	Writes a configuration parameter value into a data container.
Syntax	<code>int SetConfigPara (string id, bool data, bool EEPROM)</code> <code>int SetConfigPara (string id, byte data, bool EEPROM)</code> <code>int SetConfigPara (string id, byte[] data, bool EEPROM)</code> <code>int SetConfigPara (string id, uint data, bool EEPROM)</code> <code>int SetConfigPara (string id, long data, bool EEPROM)</code> <code>int SetConfigPara (string id, string data, bool EEPROM)</code>
Description	<p>Method for writing a configuration parameter value onto a data container.</p> <p>The parameter <i>id</i> passes the parameter name from the namespace OBID.ReaderConfig. The last parameter specifies the destination memory EEPROM (true) or RAM (false).</p> <p>If an error occurs, the method returns an error code (< 0).</p>
Return Value	Fedm.OK or an error code (< 0)
Exceptions	FedmException
Example	

7.1.42. TestConfigPara

Function	Checks a configuration parameter name for a reader type.
Syntax	int TestConfigPara (string id)
Description	<p>Method for testing a configuration parameter name for the reader type set in the reader class. If the parameter name is not supported, the method returns with Fedm.ERROR_UNSUPPORTED_NAMESPACE.</p> <p>The parameter <i>id</i> passes the parameter name from the namespace OBID.ReaderConfig. The last parameter specifies the destination memory EEPROM (true) or RAM (false).</p> <p>If an error occurs, the method returns an error code (< 0).</p>
Return Value	Fedm.OK or an error code (< 0)
Exceptions	FedmException
Example	

7.1.43. GetByteContainer

Function	Returns an array with the complete Reader configuration.
Syntax	byte[] GetByteContainer(int arrayID)
Description	<p>This method returns a byte array with the complete Reader configuration. The Reader configuration remains unchanged.</p> <p>The constants EEDATA_MEM or RAMDATA_MEM are allowed as an <i>arrayed</i>. These constants are defined in the structure Fedm.</p> <p>The byte array has a length of 1024 bytes.</p>
Return Value	A byte array with the Reader configuration or zero if there was an error.
Exceptions	FedmException
Example	

7.1.44. SetByteContainer

Function	Overwrites the complete Reader configuration in a data container.
Syntax	int SetByteContainer(int arrayID, byte[] array)
Description	<p>This method overwrites the complete Reader configuration in a data container with the contents of the byte array <i>array</i>. The Reader configuration remains unchanged.</p> <p>The constants EEDATA_MEM or RAMDATA_MEM are allowed as an <i>arrayed</i>. These constants are defined in the structure Fedm.</p> <p>The byte array has a length of 1024 bytes.</p>
Return Value	Error code (<0) or Fedm.OK
Exceptions	FedmException
Example	

7.1.45. GetTagList

Function	Returns a Dictionary with TagHandler objects.
Syntax	Dictionary<string, FedmlscTagHandler>GetTagList()
Description	This method returns the TagHandler list as a Dictionary which was previously built by the Inventory command TagInventory.
Return Value	Empty or filled Dictionary.
Exceptions	FedmException
Example	

7.1.46. GetTagHandler, GetSelectedTagHandler

Function	Returns a TagHandler objects.
Syntax	FedmlscTagHandler GetTagHandler(string uid) FedmlscTagHandler GetSelectedTagHandler()
Description	This method returns a TagHandler which was previously built by the Inventory command TagInventory or rebuilt by the Select command TagSelect.
Return Value	TagHandler or null.
Exceptions	FedmException
Example	

7.1.47. CreateNonAddressedTagHandler

Function	Creates a TagHandler object for non-addressed communication.
Syntax	FedmlscTagHandler CreateNonAddressedTagHandler(uint TagHandlerType)
Description	<p>This method creates a TagHandler object of type <i>TagHandlerType</i> for non-addressed tag communication.</p> <p>All TagHandler types are collected in the class FedmlscTagHandler, while not every TagHandler type is specified for non-addressed tag communication. More information about this communication mode can be found in the datasheet of the transponder.</p> <p>Important Note: only one TagHandler instance can be created for non-addressed tag communication. Every call of CreateNonAddressedTagHandler destroys the previously created NonAddressedTagHandler!</p>
Return Value	TagHandler or null.
Exceptions	FedmException
Example	

7.1.48. GetTableItem

Function	Returns an entry from the table.		
Syntax	GetTableItem(int idx, int tableID)		
Description	This method returns an entry from the table <i>tableID</i> which is referenced by the parameter <i>idx</i> .		
	The method can be used with the following tables:		
	TableID	BRM_TABLE	ISO_TABLE
	Support	X	X
Return Value	Table entry or zero if there was an error.		
Exceptions	FedmException		
Example			

7.1.49. SetTableItem

Function	Sets an entry in the table.		
Syntax	SetTableItem(int idx, int tableID, FedmTableItem item)		
Description	This method sets an entry in the table <i>tableID</i> which is referenced by the parameter <i>idx</i> . The transponder data are not changed.		
	The method can be used with the following tables:		
	TableID	BRM_TABLE	ISO_TABLE
	Support		X
Return Value	Fedm.OK or an error code (< 0)		
Exceptions	FedmException		
Example			

7.1.50. GetTable

Function	Returns a copy of the table.		
Syntax	FedmTableItem[] GetTable(int tableID)		
Description	This method returns a copy of the table <i>tableID</i> . The method can be used with the following tables:		
	TableID	BRM_TABLE	ISO_TABLE
	Support	X	X
Return Value	Table copy or zero if there was an error.		
Exceptions	FedmException		
Example			

7.1.51. SetTable

Function	Overwrites a table		
Syntax	int SetTableItem(FedmTableItem[] item)		
Description	This method overwrites the table <i>tableID</i> in the Reader object. The transponder data are not changed. The method can be used with the following tables:		
	TableID	BRM_TABLE	ISO_TABLE
	Support		X
Return Value	Fedm.OK or an error code (< 0)		
Exceptions	FedmException		
Example			

7.1.52. GetTableSize

Function	Returns the size of the table.		
Syntax	int GetTableSize(int tableID)		
Description	This method returns the size of the table <i>tableID</i> .		
	The method can be used with the following tables:		
	TableID	BRM_TABLE	ISO_TABLE
	Support	X	X
Return Value	Size of the table.		
Exceptions	FedmException		
Example			

7.1.53. SetTableSize

Function	Sets the size of the table.		
Syntax	(1) int SetTableSize(int tableID, int size) (2) int SetTableSize(int tableID, int size, int rxDB_BlockCount, int rxDB_BlockSize, int txDB_BlockCount, int txDB_BlockSize)		
Description	<p>This methods set the dimension of the table <i>tableID</i> to <i>size</i>. This is equivalent to the maximal number of transponders simultaneously in the RF-Field which can be handled with the tables.</p> <p>The size of the tables is not set in the constructor. Therefore it is necessary to set the table sizes to the number of expected items before first using.</p> <p>Method (1) dimensions the specified table and sets the buffer for the transponder data to 256 data blocks with each up to 32 bytes.</p> <p>Method (2) dimensions the specified table and provides to set the buffer for the transponder data to customized values.</p> <p>The methods can be used with the following tables:</p>		
	TableID	BRM_TABLE	ISO_TABLE
	Support	X ⁽¹⁾	X ^{(1) and (2)}
Return Value	Fedm.OK or an error code (< 0)		
Exceptions	FedmException		
Example			

7.1.54. GetTableLength

Function	Returns the number of entries in a table.		
Syntax	int GetTableLength(int tableID)		
Description	This method returns the number of entries in the table <i>tableID</i> . The method can be used with the following tables:		
	TableID	BRM_TABLE	ISO_TABLE
	Support	X	X
Return Value	Number of elements in the table or an error code (< 0)		
Exceptions	FedmException		
Example			

7.1.55. SetTableLength

Function	Sets the number of entries in a table.		
Syntax	int SetTableLength(int tableID, int length)		
Description	This method sets the number of entries in the table <i>tableID</i> . This method is necessary in the case of setting new table items with SetTableData or SetTableItem prior to the communication with transponders. The method can be used with the following tables:		
	TableID	BRM_TABLE	ISO_TABLE
	Support		X
Return Value	Fedm.OK or error code (< 0)		
Exceptions	FedmException		
Example			

7.1.56. ResetTable

Function	Deletes all entries from a table.		
Syntax	int ResetTable(int tableID)		
Description	This method deletes all entries from a table. The table size set with <i>SetTableSize</i> remains unchanged. After invoking the method <i>ResetTable</i> the contents of the table are deleted and the number of valid entries (TableLength) is set to 0. The method can be used with the following tables:		
	TableID	BRM_TABLE	ISO_TABLE
	Support	X	X
Return Value	Fedm.OK or an error code (< 0)		
Exceptions	FedmException		
Example			

7.1.57. GetTableData

Function	Overloaded method for reading a table value.		
Syntax	int GetTableData(int idx, int tableID, int dataID, out bool data) int GetTableData(int idx, int tableID, int dataID, out byte data) int GetTableData(int idx, int tableID, int dataID, int blockNr, out byte[] data) int GetTableData(int idx, int tableID, int dataID, out uint data) int GetTableData(int idx, int tableID, int dataID, out long data) int GetTableData(int idx, int tableID, int dataID, out string data) int GetTableData(int idx, int tableID, int dataID, int blockNr, out string data)		
Description	This method reads a value from a table of index <i>idx</i> . If an error occurs an error code is returned. For exact usage of this method, see 10.5.3. Constants for dataID . The method can be used with the following tables:		
	TableID	BRM_TABLE	ISO_TABLE
	Support	X	X
Return Value	Fedm.OK or an error code (< 0)		
Exceptions	FedmException		
Example			

7.1.58. SetTableData

Function	Overloaded method for setting a table value.		
Syntax	int SetTableData(int idx, int tableID, int dataID, bool data) int SetTableData(int idx, int tableID, int dataID, byte data) int SetTableData(int idx, int tableID, int dataID, int blockNr, byte[] data) int SetTableData(int idx, int tableID, int dataID, uint data) int SetTableData(int idx, int tableID, int dataID, long data) int SetTableData(int idx, int tableID, int dataID, string data) int SetTableData(int idx, int tableID, int dataID, int blockNr, string data)		
Description	This method sets a value in a table at <i>idx</i> . If an error occurs, an error code is returned. For exact usage of this method see 10.5.3. Constants for dataID . The method can be used with the following tables:		
	TableID	BRM_TABLE	ISO_TABLE
	Support		X
Return Value	Fedm.OK or an error code (< 0)		
Exceptions	FedmException		
Example			

7.1.59. VerifyTableDataBlocks

Function	Method for verifying the sent with the received data blocks.		
Syntax	int VerifyTableDataBlocks(int idx, int tableID, int dataID, int blockNr, int blockCnt)		
Description	The internal tables of type FedmTableItem have separate memories for received and sent transponder data. This allows verification of the sent with the received data. The table attribute blockSize, which indicates the number of bytes in each data block, is used internally. Therefore the BlockSize must have been previously set (for example by reading a data block). If the contents of the data blocks are the same, the method returns Fedm.OK, otherwise Fedm.ERROR_VERIFY. The method can be used with the following tables:		
	TableID	BRM_TABLE	ISO_TABLE
	Support		X
Return Value	Fedm.OK or an error code (< 0)		
Exceptions	FedmException		
Example			

7.1.60. FindTableIndex

Function	Overloaded method for finding a table index.		
Syntax	<code>int FindTableIndex(int startIdx, int tableID, long dataID, bool data)</code> <code>int FindTableIndex(int startIdx, int tableID, long dataID, byte data)</code> <code>int FindTableIndex(int startIdx, int tableID, long dataID, uint data)</code> <code>int FindTableIndex(int startIdx, int tableID, long dataID, long data)</code> <code>int FindTableIndex(int startIdx, int tableID, long dataID, string data)</code>		
Description	The method uses the criteria of the passed parameters to find a table entry. If a table entry was found, the method returns a null-based index, otherwise Fedm.ERROR_NO_TABLE_DATA. For exact usage of this method see 10.5.3. Constants for dataID . The method can be used with the following tables:		
	TableID	BRM_TABLE	ISO_TABLE
	Support	X	X
Return Value	Index or an error code (< 0)		
Exceptions	FedmException		
Example			

7.1.61. AddEventListener

Function	Overloaded method for reporting an event handling routine.	
Syntax	void AddEventListener (FelscListener l, int evt) void AddEventListener(FeUsbListener l, int evt)	
Description	This method is used to implement an event handling routine. AddEventListener must be separately invoked for each event listed in the table below. Each event can be added only once for each listener (the receiver object).	
	The event IDs listed in the following tables are possible:	
	FelscListener:	
	Event ID⁴	Explanation
	TRANSCIVE_STRING_EVENT	A string with date and time is sent to the listener for both the send and receive protocol.
	SEND_STRING_EVENT	A string with date and time is sent to the listener for the send protocol.
	RECEIVE_STRING_EVENT	A string with date and time is sent to the listener for the receive protocol.
	SCANNER_PRT_EVENT	A byte array with the receive data is sent to the listener. The reporting of this event starts internally a continuous receive process for data which a Reader outputs in scan mode.
	SEND_PRT_EVENT	A byte array with the send data is sent to the listener.
	RECEIVE_PRT_EVENT	A byte array with the receive data is sent to the listener.

⁴ see FelscListenerConst structure

	FeUsbListener:	
	Event ID⁵	Explanation
	FEUSB_CONNECT_EVENT	The new connection of a USB Reader is reported to the listener.
	FEUSB_DISCONNECT_EVENT	Disconnection of a USB Reader is reported to the listener.
Exceptions	FedmException	
Cross-reference	7.6.FelscListener , 7.7.FeUsbListener	
Example	<pre> using System; using OBID; class MyClass : FeIscListener { // Overloaded methods from FeIscListener // are invoked for events public void OnSendProtocol(FedmIscReader reader, string sendProtocol) { Console.WriteLine(sendProtocol); } void OnSendProtocol(FedmIscReader reader, byte[] sendProtocol) { // at this point a handling method for the byte array can // be implemented } public void OnReceiveProtocol(FedmIscReader reader, string receiveProtocol) { Console.WriteLine(receiveProtocol); } void OnReceiveProtocol(FedmIscReader reader, byte[] receiveProtocol) { // at this point a handling method for the byte array can // be implemented } MyClass() { try { int readerType; this.reader = new FedmIscReader(); reader.ConnectCOMM(1); // Open connection to COM Reader reader.FindBaudRate(); // Setting the baud rate // Reporting the event handling routines reader.AddEventListener(this, FeIscListenerConst.SEND_STRING_EVENT); reader.AddEventListener(this, FeIscListenerConst.RECEIVE_STRING_EVENT); reader.SendProtocol(0x65); // Get the SoftwareVersion // Note! The protocols are displayed! readerType = reader.GetReaderType(); // Get Reader type Console.WriteLine("Reader type: "); } catch { } } } </pre>	

⁵ see FeUsbListenerConst structure

```
        Console.WriteLine(readerType);
        Console.ReadLine();
        // Removing the event handling routines
        reader.RemoveEventListener(this, FeIscListenerConst.SEND_STRING_EVENT);
        reader.RemoveEventListener(this, FeIscListenerConst.RECEIVE_STRING_EVENT);
    }
    catch (FedmException e)
    {
        Console.WriteLine(e.Message);
    }
    catch (FePortDriverException e)
    {
        Console.WriteLine(e.Message);
    }
    catch (FeReaderDriverException e)
    {
        Console.WriteLine(e.Message);
    }
}

/// <summary>
/// The main entry point for the application.
/// </summary>
[STAThread]
static void Main(string[] args)
{
    new MyClass();
}

public FedmIscReader reader;
}
```

7.1.62. RemoveEventListener

Function	Removes a previously installed event handling routine.
Syntax	void RemoveEventListener(FelscListener l, int evt) void RemoveEventListener(FeUsbListener l, int evt)
Description	This method is used to remove an even handling routine. For additional information see AddEventListener .
Exceptions	FedmException
Example	A detailed example can be found in the description for the method AddEventListener .

7.1.63. StartAsyncTask

Function	An inventory or notification task is started asynchronous to the application		
Syntax	int StartAsyncTask(int TaskID, FedmTaskListener listener, FedmTaskOption opt)		
Description	<p>This method starts an asynchronous task. An asynchronous task is an internal thread which e.g. sends an inventory command to the reader and waits for the reply for a time up to the timeout. Signaling of the reply data or the cancel condition to the application is done by invoking a delegate.</p> <p>The task behavior is specified in the parameter <i>iTaskID</i>. Three tasks are currently defined:</p>		
	Task	TaskID	Remarks
	One-time Inventory	ID_FIRST_NEW_TAG	<p>A task can only started if the following option is integrated in the Reader's firmware: the Reader protocol [0xB0][0x01] Inventory must support an optional NOTIFY flag in its Mode byte.</p> <p>After receiving the Reader protocol within the specified time, the task invokes the delegate OnNewTag and automatically closes itself. If the time is exceeded, the delegate is invoked and the status 0x01 (No transponder in read field) send and the task ended. In case of error the task is always ended immediately and the delegate transmits the error code.</p> <p>Serial, USB and TCP/IP interfaces are supported, whereby the ports must be open before starting the task. Autonomous opening of the connection via TCP/IP by the Reader or a suitable converter for sending the data is not possible.</p>
	Repeating Inventory	ID EVERY_NEW_TAG	<p>The same conditions as for one-time inventory apply, with the following difference:</p> <p>Repeating inventory defines a cyclical task which can only be cancelled by CancelAsyncTask. A cycle corresponds to a one-time inventory and ends on a wait loop until the next cycle has been triggered by the application using TriggerAsyncTask. Application-side triggering ensures that an application has time for receiving and processing the inventory data.</p>

	Receiving notifications	ID_NOTIFICATION	<p>A task should only be started if notification mode is integrated and activated in the Reader's firmware. Only TCP/IP communication is supported. Possible connection options are (see system manual for the Reader):</p> <ul style="list-style-type: none"> - Temporary opening of the connection by the Reader for the duration of data transmission - Continuous opening of the connection by the Reader (in development) - Continuous opening of the connection by the host (in development) <p>The task defines an endless task which can only be cancelled using CancelAsyncTask or in case of error during the initialization phase is ended immediately after invoking the delegate OnNewNotification.</p> <p>The task waits for reception of the Buffered-Read-Mode data and then invokes the delegate OnNewNotification. After the delegate returns, data can immediately be received again by the Reader.</p> <p>In case of transmission errors the delegate is invoked with the error code and the receiving procedure then resumed. If the Keep-Alive option is activated (by default), then the listener socket is closed automatically after a break of the network cable or after loss of power and is recovered again. This ensures the reliability of the network connection.</p> <p>The timing for closing the socket by Keep-Alive is calculated by the formula:</p> $\text{IdleTime} + \text{RepeatCount} * \text{IntervalTime}$ <p>where the default settings are:</p> <ul style="list-style-type: none"> - IdleTime is 500ms - RepeatCount is 5 (Window XP) or 10 (Vista or 7) - IntervalTime is 500ms <p>Note: Depending on the Reader setting large quantities of data may be sent by the Reader in very short time intervals. Without use of a handshake procedure (see system manual for the Reader) data may be lost if the host is not appropriate for the quantity of notifications.</p>
	All the data relevant to the delegate are contained in the property class FedmTaskOption.		
Return Value	Fedm.OK or an error code (< 0).		
Exceptions	FedmException		
Example (VB.NET)	<pre>taskOpt = New FedmTaskOption() taskOpt.IPPort = "192.168.1.1" taskOpt.NotifyWithAck = 0 taskOpt.Timeout = 1000 ' 1000 ms reader.StartAsyncTask(FedmTaskOption.ID_NOTIFICATION, Me, taskOpt)</pre>		

7.1.64. CancelAsyncTask

Function	Cancels an inventory or notification task.
Syntax	int CancelAsyncTask()
Description	<p>This function cancels an asynchronous task.</p> <p>You should not normally use one-time inventory (started with TaskID = ID_FIRST_NEW_TAG) to quit this function. You should end repeating inventory (started with TaskID = ID_EVERY_NEW_TAG) using this function if the delegate was ended and the internal thread is waiting for the next trigger. This ensures that the task in the reader is ended and it can again process reader tasks.</p> <p>Notification tasks must always be canceled with this function.</p> <p>The cancellation of the task is locked if the task execution is just inside the delegate. This prevents deadlocks. In this case this method returns directly with the return value FEISC_ERR_TASK_BUSY (-4084) and the application must invoke CancelAsyncTask until the return value is not -4084. On application-side the return from the callback function must be guaranteed.</p>
Return Value	Fedm.OK or an error code (< 0).
Exceptions	FedmException
Example	

7.1.65. TriggerAsyncTask

Function	Triggers the next cycle in the inventory task.
Syntax	void TriggerAsyncTask()
Description	<p>This function is used to trigger the next inventory cycle in the asynchronous task. The asynchronous task must have been previously started with the TaskID = ID_EVERY_NEW_TAG.</p> <p>This method is always invoked after the delegate has been exited. Without this invoke a task with repeating function hangs up in a wait loop.</p>
Exceptions	FedmException
Example	

7.2. FedmIsoTableItem, FedmBrmTableItem

7.2.1. Data members in FedmISOTableItem

The following table lists all (public and private) data members. The direct access to the public members is possible. All private members have to be queried or set with the methods *GetData* or *SetData*.

More information about each data member can be found in the system manual of each OBID®-Reader.

Data Member	Relevance	Description
Category: Transponder specific information		
uid[]	all	Serial Number of Transponder
transponderType	all	Type of Transponder (s. appendix in system manual of OBID®-Reader)
DsflD	ISO 15693	Data Storage Format Identifier
trInfo	ISO 14443-A	Transponder Information
optInfo	ISO 14443-A	Optional Information
protolInfo	ISO 14443-B	Protocol Info Byte
chipID	STM SR176 and SRIxx (ISO 14443 classic-pro reader)	See system manual of OBID®-Reader
IDDT	EPC Class1 Gen2, ISO 18000-3M3	Identifier Data Type (see system manual of OBID®-Reader)
class1Gen2PC[]	EPC Class1 Gen2, ISO 18000-3M3	See system manual of OBID®-Reader
AFI	ISO 15693	Application Family Identifier
memSize	ISO 15693	Memory Size
ICRef	ISO 15693	IC Referenz
FSCI	ISO 14443-4	See system manual of OBID®-Reader
FWI	ISO 14443-4	See system manual of OBID®-Reader
DSI	ISO 14443-4	See system manual of OBID®-Reader
DRI	ISO 14443-4	See system manual of OBID®-Reader
NAD	ISO 14443-4	See system manual of OBID®-Reader
CID	ISO 14443-4	See system manual of OBID®-Reader
productCode	ISO 14443-4 ASK CTx	See system manual of OBID®-Reader
fabCode	ISO 14443-4 ASK CTx	See system manual of OBID®-Reader
appCode	ISO 14443-4 ASK CTx	See system manual of OBID®-Reader
embedderCode	ISO 14443-4 ASK CTx	See system manual of OBID®-Reader
verlog	Innovatron (ISO 14443B')	See system manual of OBID®-Reader
config	Innovatron (ISO 14443B')	See system manual of OBID®-Reader
atr[]	Innovatron (ISO 14443B')	See system manual of OBID®-Reader
challenge	EPC Class1 Gen2: NXP UCODE DNA	See system manual of OBID®-Reader
cryptoBlock	EPC Class1 Gen2: NXP UCODE DNA	See system manual of OBID®-Reader

Category: Data of Transponder			
blockSize	all	Block size (number of bytes in each data block)	
securityStatus[]	ISO 15693	Status information for each data block	Access with GetData() and SetData()
rxPubData[]	all	Data blocks after read	
txPubData[]	all	Data blocks before write	
rxDB_EpcBank[]	EPC Class1 Gen2, ISO 18000-3M3	Data blocks after read	
txDB_EpcBank[]	EPC Class1 Gen2, ISO 18000-3M3	Data blocks before write	
rxDB_TidBank[]	EPC Class1 Gen2, ISO 18000-3M3	Data blocks after read	
txDB_TidBank[]	EPC Class1 Gen2, ISO 18000-3M3	Data blocks before write	
rxDB_ResBank[]	EPC Class1 Gen2, ISO 18000-3M3	Data blocks after read	
rxDB_ResBank[]	EPC Class1 Gen2, ISO 18000-3M3	Data blocks before write	
Category: Informationen about Antennas and Field Measurements			
antCount	ISO 15693, EPC Class1 Gen2, ISO 18000-3M3	Number of antennas, from where the tags are read	Access with GetRSSI()
antNumber[]	ISO 15693, EPC Class1 Gen2, ISO 18000-3M3	Array with antenna numbers	
antStatus[]	ISO 15693, EPC Class1 Gen2, ISO 18000-3M3	Array with status information	
antRSSI[]	ISO 15693, EPC Class1 Gen2, ISO 18000-3M3	Array with RSSI measured value	
Category: Data integrity			
isUid	all	Signals, that uid and transponderType are valid	
isAFI	ISO 15693	Signals, that AFI is valid	
isSysInfo	ISO 15693	Signals, that ISO 15693 system info is valid	
isISO14443Info	ISO 14443	Signals, that ISO 14443 system info is valid	
isRSSI	ISO 15693, EPC Class1 Gen2, ISO 18000-3M3	Signals, that antenna info and RSSI measured values are valid	
isBlockSizeSet	all	Signals, that blockSize is valid	
isSelected	ISO 15693, ISO 14443	Signals, that this table item contains the selected Transponder	

7.2.2. Data members in FedmBRMTableItem

The following table lists all (public and private) data members. The direct access to the public members is possible. All private members have to be queried or set with the methods *GetData* or *SetData*.

More information about each data member can be found in the system manual of each OBID®-Reader.

Data Member	Relevance	Description	
Category: Transponder specific information			
uid[]	all	Serial Number of Transponder	
transponderType	all	Type of Transponder (s. appendix in system manual of OBID®-Reader)	
trInfo	ISO 14443-A	Transponder Information	
IDDT	EPC Class1 Gen2, ISO 18000-3M3	Identifier Data Type (see system manual of OBID®-Reader)	
class1Gen2PC[]	EPC Class1 Gen2, ISO 18000-3M3	See system manual of OBID®-Reader	
AFI	ISO 15693	Application Family Identifier	
DsflID	ISO 15693	Data Storage Format Identifier	
Category: Data of Transponder			
blockSize	all	Block size (number of bytes in each data block)	
dbAddress	all	Start address	
blockCount	all	Number of data blocks	
rxPubData[]	all	Data blocks after read	Access with GetData()
Category: Informationen about Antennas and Field Measurements			
antennaNumber	all	Flag-Field signaling, from which antenna the tag is read. Alternative to antNumber[] and antRSSI[]	
antCount	ISO 15693, EPC Class1 Gen2, ISO 18000-3M3	Number of antennas, from where the tags are read	Alternative to antennaNumber Access with GetRSSI()
antNumber[]	ISO 15693, EPC Class1 Gen2, ISO 18000-3M3	Array with antenna numbers	
antRSSI[]	ISO 15693, EPC Class1 Gen2, ISO 18000-3M3	Array with RSSI measured value	
Category: Miscellaneous information			
input	all	Flag-Field signaling Input events	
status	all	Status Information about the read process	
readerTime	all	Date and Time in type FeliscReaderTime	
macAddr	all	MAC-Address	
direction	Gate People Counter	Direction information	

Category: Data integrity		
isUid	all	Signals, that uid and transponderType are valid
isAntNr		Signals, that antenna information in antennaNumber is valid
isRSSI	ISO 15693, EPC Class1 Gen2, ISO 18000-3M3	Signals, that RSSI measured values are valid
isDB	all	Signals, that data blocks are read
isTimer	all	Signals, that Time is valid
isDate	all	Signals, that Date is valid
isInput	all	Signals, that Input event information is valid
isMacAddr	all	Signals, that the MAC-Address is valid
isDirection	Gate People Counter	Signals, that the direction information is valid

7.2.3. GetData

Function	Overloaded method for reading a table value		
Syntax	int GetData(int dataID, out bool data) int GetData(int dataID, out byte data) int GetData(int dataID, int blockNr, out byte[] data) int GetData(int dataID, out uint data) int GetData(int dataID, out long data) int GetData(int dataID, out string data) int GetData(int dataID, int blockNr, out string data)		
Description	This method reads a value from the table. If an error occurs, an error code is returned. For exact usage of this method see 10.5.3. Constants for dataID . The method can be used with the following tables:		
	tableID	BRM_TABLE	ISO_TABLE
	Support	X	X
Return Value	Fedm.OK or an error code (< 0)		
Exceptions	None		
Example			

7.2.4. SetData

Function	Overloaded method for setting a table value.		
Syntax	int SetData(int dataID, bool data) int SetData(int dataID, byte data) int SetData(int dataID, int blockNr, byte[] data) int SetData(int dataID, uint data) int SetData(int dataID, long data) int SetData(int dataID, string data) int SetData(int dataID, int blockNr, string data)		
Description	This method sets a value in the table. If an error occurs, an error code is returned. For exact usage of this method see 10.5.3. Constants for dataID . The method can be used with the following tables:		
	tableID	BRM_TABLE	ISO_TABLE
	Support		X
Return Value	Fedm.OK or an error code (< 0)		
Exceptions	None		
Example			

7.2.5. GetRSSI

Function	Method returns RSSI values		
Syntax	Dictionary<byte, FedmlscRssItem> GetRSSI()		
Description	This method returns a dictionary with RSSI value from the table. The key is the antenna number and the value an object with the RSSI value. The method can be used with the following tables:		
	tableID	BRM_TABLE	ISO_TABLE
	Support	X	X
Return Value	Dictionary with RSSI values or in error case an empty dictionary		
Exceptions	None		
Example			

7.2.6. VerifyDataBlocks

Function	Method for verifying sent with received data blocks.		
Syntax	int VerifyDataBlocks(int blockNr, int blockCnt)		
Description	Tables of type FedmTableItem have separate memories for received and sent transponder data. This allows verification of the sent with the received data. The table attribute blockSize , which indicates the number of bytes in each data block, is used internally. Therefore the BlockSize must have been previously set (for example by reading a data block). If the contents of the data blocks are the same, the method returns Fedm.OK , otherwise Fedm.ERROR_VERIFY The method can be used with the following tables:		
	tableID	BRM_TABLE	ISO_TABLE
	Support		X
Return Value	Fedm.OK or an error code (< 0).		
Exceptions	None		
Example			

7.2.7. IsDataValid

Function	Method for checking the validity of table data.
Syntax	bool IsDataValid(int dataID)
Description	Overwritten method for checking a data element for validity.
Return Value	True if the data value is valid.
Exceptions	None
Example	

7.2.8. GetIdentifier

Function	Method for identifying a table.
Syntax	string GetIdentifier()
Description	The method returns a string object with the table identifier.
Return Value	„ISO“ or „BRM“
Exceptions	None
Example	

7.3. FedmlscFunctionUnit

7.3.1. FedmlscFunctionUnit

Function	Generates a new instance of the class FedmlscFunctionUnit (constructor).
Syntax	FedmlscFunctionUnit(FedmlscReader, int FUType)
Description	<p>A function unit object is generated. Only a function unit object can be used to run the protocol functions.</p> <p>The constructor needs an instance of a reader object and the type of the function unit.</p>
Return Value	If a function unit object was able to be created without error, the new instance of the class FedmlscFunctionUnit is returned.
Exceptions	In case of error the method throws the exception FedmException .
Example	<pre>using OBID; ... try { fu = new FedmlscFunctionUnit(reader, FedmlscFunctionUnit.FU_TYPE_MUX); } catch (FedmException e) { Console.WriteLine("Exception when generating a new object: " + e.Message); } private FedmlscReader reader; private FedmlscFunctionUnit fu;</pre>

7.3.2. GetFUType

Function	Returns the type number of a function unit.
Syntax	int GetFUType()
Description	Returns the type number of a function unit. The type numbers are members of the class FedmFunctionUnit.
Return Value	Type number.
Exceptions	None
Example	

7.3.3. GetLastError

Function	Returns the last error code.
Syntax	int GetLastError()
Description	Returns the last error code.
Return Value	Error code (<0)
Exceptions	None

7.3.4. GetErrorText

Function	Gets an error text.
Syntax	string GetErrorText(int errorCode)
Description	Returns a short error text referenced by the error code.
Return Value	An error text
Exceptions	None

7.3.5. SendProtocol

Function	The central communications method.
Syntax	int SendProtocol(byte cmd)
Description	Protocol traffic is carried out using the method <i>SendProtocol</i> . This will pass only the control byte for the selected protocol. All data needed for the protocol transfer are taken from the data container TmpData. Therefore you must ensure that all required parameters have been previously set.
Return Value	An integer with the error code (< 0) or OK (0) or Statusbyte of Reader response (>0)
Exceptions	In case of error the method throws one of the exceptions FedmException , FePortDriverException , FeReaderDriverException .
Example	For a detailed description of how to use this method, see

7.3.6. GetData

Function	Gets a datum from a data container
Syntax	<code>int GetData(string id, out bool data)</code> <code>int GetData(string id, out byte data)</code> <code>int GetData(string id, out uint data)</code> <code>int GetData(string id, out long data)</code> <code>int GetData(string id, out string data)</code>
Description	Method for reading a value from a data container. The storage location of the datum is determined by the parameter <i>id</i> . If an error occurs, the method returns an error code (< 0).
Return Value	Fedm.OK or an (< 0)
Exceptions	FedmException
Example	

7.3.7. SetData

Function	Writes a datum into a data container
Syntax	<code>int SetData(string id, bool data)</code> <code>int SetData(string id, byte data)</code> <code>int SetData(string id, uint data)</code> <code>int SetData(string id, long data)</code> <code>int SetData(string id, string data)</code>
Description	Method for setting a value in a data container. The storage location of the datum is determined by the parameter <i>id</i> . If an error occurs, the method returns an error code (< 0).
Return Value	Fedm.OK or an error code (< 0)
Exceptions	FedmException
Example	

7.3.8. AddChild

Function	Add a function unit as a child object to the parents child list.
Syntax	int AddChild(int outNr, FedmIsFunctionUnit child)
Description	Add a function unit <i>child</i> to the parents child list at output <i>outNr</i> . A previously saved link to another child object at the same outout number <i>outNr</i> will be overwritten.
Return Value	Fedm.OK or an error code (< 0)
Exceptions	FedmException
Example	

7.3.9. DeleteChild

Function	Remove of a function unit from the parents child list.
Syntax	int DeleteChild(int outNr)
Description	Remove of a function unit from the parents child list at output number <i>outNr</i> .
Return Value	Fedm.OK or an error code (< 0)
Exceptions	FedmException
Example	

7.3.10. GetChild

Function	Return of a function unit from parents child list.
Syntax	int GetChild(int outNr)
Description	Return of a function unit from parents child list at output <i>outNr</i> .
Return Value	Fedm.OK or an error code (< 0)
Exceptions	FedmException
Example	

7.4. FedmlscPeopleCounter

7.4.1. GetCounterValues

Function	Query of counter values										
Syntax	long[] GetCounterValues()										
Description	This method queries all four counter values from a People Counter										
Return Value	<p>Array with four counter values.</p> <table border="1"> <thead> <tr> <th>Index</th><th>Counter</th></tr> </thead> <tbody> <tr> <td>0</td><td>Counter 1 of Radar Detector 1</td></tr> <tr> <td>1</td><td>Counter 2 of Radar Detector 1</td></tr> <tr> <td>2</td><td>Counter 1 of Radar Detector 2</td></tr> <tr> <td>3</td><td>Counter 2 of Radar Detector 2</td></tr> </tbody> </table>	Index	Counter	0	Counter 1 of Radar Detector 1	1	Counter 2 of Radar Detector 1	2	Counter 1 of Radar Detector 2	3	Counter 2 of Radar Detector 2
Index	Counter										
0	Counter 1 of Radar Detector 1										
1	Counter 2 of Radar Detector 1										
2	Counter 1 of Radar Detector 2										
3	Counter 2 of Radar Detector 2										
Exceptions	In case of error the method throws one of the exceptions FedmException , FePortDriverException , FeReaderDriverException .										
Example	<pre> FedmlscReader reader; ... // connecting with internal ReadReaderInfo() reader.ConnectTCP("192.168.10.10", 10001); ... long[] values = null; FedmlscPeopleCounter pc = null; // query dictionary with People Counter objects Dictionary<byte, FedmlscPeopleCounter> mapPC = reader.GetPeopleCounterMap(); // get People Counter object with busaddress 1 bool ret = mapPC.TryGetValue(1, out pc); if (ret) { try { // query counter values values = pc.GetCounterValues(); } catch (Exception ex) { // catch an error case } } </pre>										

7.4.2. SetCounterValues

Function	Set of counter
Syntax	<code>int SetCounterValues(long radar1Counter1, long radar1Counter2, long radar2Counter1, long radar2Counter2)</code>
Description	This method sets all counter values to the transferred value.
Return Value	An integer with the error code (< 0) or OK (0) or Statusbyte of response (>0)
Exceptions	In case of error the method throws one of the exceptions FedmException , FePortDriverException , FeReaderDriverException .
Example	<pre>FedmlscReader reader; ... // connecting with internal ReadReaderInfo() reader.ConnectTCP("192.168.10.10", 10001); ... int status =0; FedmlscPeopleCounter pc = null; // query dictionarywith People Counter objects Dictionary<byte, FedmlscPeopleCounter> mapPC = reader.GetPeopleCounterMap(); // get People Counter object with busaddress 1 bool ret = mapPC.TryGetValue(1, out pc); if (ret) { try { // set all counter values to zero status = pc.SetCounterValues(0, 0, 0, 0); } catch (Exception ex) { // catch an error case } }</pre>

7.4.3. SetOutputsOn

Function	Set of digital outputs
Syntax	<pre>int SetOutputsOn(int holdTime1, int holdTime2, int holdTime3)</pre>
Description	<p>This method sets up to three digital outputs for the activation time holdTimeX. A holdTimeX value of 0 has no effect for the specified output. A holdTimeX value of 65535 sets the specified output permanently.</p> <p>The value range of holdTimeX is 0...65535, while the activation time is set in steps of 100ms.</p>
Return Value	An integer with the error code (< 0) or OK (0) or Statusbyte of response (>0)
Exceptions	In case of error the method throws one of the exceptions FedmException , FePortDriverException , FeReaderDriverException .
Example	<pre>FedmlscReader reader; ... // connecting with internal ReadReaderInfo() reader.ConnectTCP("192.168.10.10", 10001); ... int status = 0; FedmlscPeopleCounter pc = null; // query dictionary with People Counter objects Dictionary<byte, FedmlscPeopleCounter> mapPC = reader.GetPeopleCounterMap(); // get People Counter object with busaddress 1 bool ret = mapPC.TryGetValue(1, out pc); if (ret) { try { // set output 1 for 1 second // set output 3 for 2 seconds status = pc.SetOutputsOn(10, 0, 20); } catch (Exception ex) { // catch an error case } }</pre>

7.4.4. SetOutputsOff

Function	Reset of digital outputs
Syntax	<code>int SetOutputsOff(bool off1, bool off2, bool off3)</code>
Description	This method resets up to three previously activated digital outputs.
Return Value	An integer with the error code (< 0) or OK (0) or Statusbyte of response (>0)
Exceptions	In case of error the method throws one of the exceptions FedmException , FePortDriverException , FeReaderDriverException .
Example	<pre>FedmlscReader reader; ... // connecting with internal ReadReaderInfo() reader.ConnectTCP("192.168.10.10", 10001); ... int status =0; FedmlscPeopleCounter pc = null; // query dictionarywith People Counter objects Dictionary<byte, FedmlscPeopleCounter> mapPC = reader.GetPeopleCounterMap(); // get People Counter object with busaddress 1 bool ret = mapPC.TryGetValue(1, out pc); if (ret) { try { // reset output 1 and 3 status = pc.SetOutputsOff(true, false, true); } catch (Exception ex) { // catch an error case } }</pre>

7.4.5. SetOutputsFlashing

Function	Set of digital outputs
Syntax	<pre>int SetOutputsFlashing(int frequencyOut1, int holdTime1, int frequencyOut2,int holdTime2, int frequencyOut3,int holdTime3)</pre>
Description	<p>This method sets up to three digital outputs for the activation time holdTimeX. Each specified output is flashing with the frequencyX. A holdTimeX value of 0 has no effect for the specified output. A holdTimeX value of 65535 sets the specified output permanently.</p> <p>The value range of holdTimeX is 0...65535, while the activation time is set in steps of 100ms.</p> <p>The value range for frequency is 1, 2, 4, 8 and represents the flashing frequency in Hz.</p>
Return Value	An integer with the error code (< 0) or OK (0) or Statusbyte of response (>0)
Exceptions	In case of error the method throws one of the exceptions FedmException , FePortDriverException , FeReaderDriverException .
Example	<pre>FedmlscReader reader; ... // connecting with internal ReadReaderInfo() reader.ConnectTCP("192.168.10.10", 10001); ... int status =0; FedmlscPeopleCounter pc = null; // query dictionary with People Counter objects Dictionary<byte, FedmlscPeopleCounter> mapPC = reader.GetPeopleCounterMap(); // get People Counter object with busaddress 1 bool ret = mapPC.TryGetValue(1, out pc); if (ret) { try { // output 1 with 1Hz for 1 second // output 3 with 2Hz for 2 second status = pc.SetOutputsFlashing(1, 10, 0, 0, 2, 20); } catch (Exception ex) { // catch an error case } }</pre>

7.5. FeHexConvert

Useful functions for converting data and editing the access constants are summarized in the class **FeHexConvert**. All functions are static and can therefore be used from any context.

7.5.1. ByteArrayToHexStringWithSpaces

Function	Conversion function.
Syntax	static System.String ByteArrayToHexStringWithSpaces(byte[] in)
Description	The function converts a byte array into a character string. Each byte is converted into two hex characters (a-f, A-F, 0-9) and a space is inserted between each two characters as a separator.
Return Value	String
Exceptions	None
Example	Pass : 0x11, 0x22, 0xF0, 0x5E Return : "11 22 F0 5E"

7.5.2. ByteArrayToHexString

Function	Overloaded converting function
Syntax	static System.String ByteArrayToHexString(byte[] in) static string ByteArrayToHexString(byte[] in, int start, int count)
Description	The first function converts a byte array into a character string. Each byte is converted into two hex characters (a-f, A-F, 0-9). The second function takes only the <i>count</i> byte starting at index <i>start</i> for the conversion. The first function is the inverse function of HexStringToByteArray .
Return Value	String
Exceptions	None
Example	Pass : 0x11, 0x22, 0xF0, 0x5E Return : "1122F05E"

7.5.3. ByteToHexString

Function	Conversion function
Syntax	static string ByteToHexString(byte in)
Description	The function converts the byte into a string with two hex characters (a-f, A-F, 0-9). Inverse of HexStringToByte .
Return Value	String
Exceptions	None
Example	Pass : 0x11 Return : "11"

7.5.4. IntegerToHexString

Function	Conversion function
Syntax	static string IntegerToHexString(int in)
Description	The function converts the int-value into a string, whereby each byte of the integer is converted into two hex characters each (a-f, A-F, 0-9). Inverse of HexStringToInteger .
Return Value	String
Exceptions	None
Example	Pass : 287502430 Return : "1122F05E"

7.5.5. LongToHexString

Function	Conversion function
Syntax	static string LongToHexString(long in)
Description	The function converts the long-value into a string, whereby each byte of the long is converted into two hex characters (a-f, A-F, 0-9). Inverse of HexStringToLong .
Return Value	String
Exceptions	None
Example	Pass : 1234813534658031710 Return : "1122F05E1122F05E"

7.5.6. HexStringToByte

Function	Conversion function
Syntax	static byte HexStringToByte(string str)
Description	The function converts a string consisting of the hex characters 0-9, a-f, A-F into an int-value. The string must have an even number of characters and may consist of a maximum of 8 characters. Inverse of ByteToHexString .
Return Value	Byte-value
Exceptions	System.ArgumentException
Example	Pass : "11" Return : 0x11

7.5.7. HexStringToByteArray

Function	Overloaded conversion function
Syntax	static byte[] HexStringToByteArray(String str) static byte[] HexStringToByteArray(string str, int start, int count)
Description	The first function converts a string into a byte array. Each two hex characters (a-f, A-F, 0-9) are converted into a byte. The second function takes only the <i>count</i> character beginning with <i>start</i> for the conversion. The first function is the inverse of HexStringToByteArray .
Return Value	Byte-Array
Exceptions	System.ArgumentException
Example	Pass : "1122F05E" Return : 0x11, 0x22, 0xF0, 0x5E

7.5.8. HexStringToInteger

Function	Conversion function
Syntax	static int HexStringToInteger(string str)
Description	The function converts a string consisting of the hex characters 0-9, a-f, A-F into an int-value. The string must contain an even number of characters and may consist of no more than 8 characters. Inverse of IntegerToHexString .
Return Value	int-Wert
Exceptions	System.ArgumentException
Example	Pass : "1122F05E" Return : 287502430

7.5.9. HexStringToLong

Function	Conversion function
Syntax	static long HexStringToLong(string str)
Description	The function converts a string consisting of the hex characters 0-9, a-f, A-F into a long value. The string must contain an even number of characters and may consist of no more than 16 characters. Inverse of LongToHexString .
Return Value	long-value
Exceptions	System.ArgumentException
Example	Pass : "1122F05E1122F05E" Return : 1234813534658031710

7.5.10. isHexString

Function	Test function for string
Syntax	static bool isHexString(string str)
Description	The function checks whether the passed string consists only of the hex characters a-f, A-F and 0-9.
Return Value	True if only the above indicated characters are contained, otherwise False.
Exceptions	None
Example	

7.5.11. GetMemIDOfID

Function	Function for access constants
Syntax	static int GetMemIDOfID(string ID)
Description	The function returns the memory ID from the access constant <i>ID</i> .
Return Value	int
Exceptions	None
Cross-reference	5.6.2.Access constants
Example	Pass : FEDM_ISC_EE_COM_BUSADR ("03 03 01 00 01 00 00") Return : 3

7.5.12. GetByteCntOfID

Function	Function for access constant
Syntax	static int GetByteCntOfID(string ID)
Description	The function returns the number of bytes making up the parameter from the access constant <i>ID</i> .
Return Value	int
Exceptions	None
Cross-reference	5.6.2.Access constants
Example	Pass : FEDM_ISC_EE_COM_BUSADR ("03 03 01 00 01 00 00") Return : 1

7.5.13. GetAdrOfID

Function	Function for access constant
Syntax	static int GetAdrOfID(string ID)
Description	The function returns the start address (index in the data container) for a parameter from the access constant <i>ID</i> .
Return Value	int
Exceptions	None
Cross-reference	5.6.2.Access constants
Example	Pass : FEDM_ISC_EE_COM_BUSADR ("03 03 01 00 01 00 00") Return : 1

7.6.FelscListener

7.6.1. OnSendProtocol

Function	Overloaded event methods for events						
Syntax	void OnSendProtocol(FedmlscReader reader, byte[] sendProtocol); void OnSendProtocol(FedmlscReader reader, string sendProtocol);						
Description	<p>These methods are invoked when you have used the AddEventHandler method of the Reader class FedmlscReader to report events that are related to a send protocol. One of the overloaded methods is invoked for the following events:</p> <table><tr><td>TRANSCIVE_STRING_EVENT</td><td>OnSendProtocol(..., string sendProtocol)</td></tr><tr><td>SEND_STRING_EVENT</td><td>OnSendProtocol(..., string sendProtocol)</td></tr><tr><td>SEND_PRT_EVENT</td><td>OnSendProtocol(..., byte[] sendProtocol)</td></tr></table>	TRANSCIVE_STRING_EVENT	OnSendProtocol(..., string sendProtocol)	SEND_STRING_EVENT	OnSendProtocol(..., string sendProtocol)	SEND_PRT_EVENT	OnSendProtocol(..., byte[] sendProtocol)
TRANSCIVE_STRING_EVENT	OnSendProtocol(..., string sendProtocol)						
SEND_STRING_EVENT	OnSendProtocol(..., string sendProtocol)						
SEND_PRT_EVENT	OnSendProtocol(..., byte[] sendProtocol)						
Return Value	None						
Exceptions	None						
Example	see example in 7.1.60. AddEventListener						

7.6.2. OnReceiveProtocol

Function	Overloaded event methods for events						
Syntax	void OnReceiveProtocol(FedmlscReader reader, byte[] receiveProtocol); void OnReceiveProtocol(FedmlscReader reader, string receiveProtocol);						
Description	<p>These methods are invoked when you have used the AddEventHandler method of the Reader class FedmlscReader to report events that are related to a receive protocol. One of the overloaded methods is invoked for the following events:</p> <table><tr><td>TRANSCIVE_STRING_EVENT</td><td>OnReceiveProtocol(..., string receiveProtocol)</td></tr><tr><td>RECEIVE_STRING_EVENT</td><td>OnReceiveProtocol(..., string receiveProtocol)</td></tr><tr><td>RECEIVE_PRT_EVENT</td><td>OnReceiveProtocol(..., byte[] receiveProtocol)</td></tr></table>	TRANSCIVE_STRING_EVENT	OnReceiveProtocol(..., string receiveProtocol)	RECEIVE_STRING_EVENT	OnReceiveProtocol(..., string receiveProtocol)	RECEIVE_PRT_EVENT	OnReceiveProtocol(..., byte[] receiveProtocol)
TRANSCIVE_STRING_EVENT	OnReceiveProtocol(..., string receiveProtocol)						
RECEIVE_STRING_EVENT	OnReceiveProtocol(..., string receiveProtocol)						
RECEIVE_PRT_EVENT	OnReceiveProtocol(..., byte[] receiveProtocol)						
Return Value	None						
Exceptions	None						
Example	see example in 7.1.60. AddEventListener						

7.7.FeUsbListener

7.7.1. OnConnectReader

Function	Event method for event
Syntax	void OnConnectReader(int deviceHandle, long deviceID);
Description	<p>This method is invoked if you have reported the event <code>FEUSB_CONNECT_EVENT</code> using the method <code>AddEventHandler</code> of the Reader class <code>FedmlscReader</code> and a USB Reader is connected to the PC.</p> <p>This procedure is very useful for telling an application that the USB Reader is available.</p> <p>In the first pass parameter a device handle is passed. This has meaning only if you are administering multiple USB Readers using the class <code>FeUsb</code>.</p> <p>The second pass parameter contains the serial number of the USB Reader.</p>
Return Value	None
Exceptions	None
Example	see equivalent example in 7.1.60. AddEventListener

7.7.2. OnDisconnectReader

Function	Event method for event
Syntax	void OnDisconnectReader(int deviceHandle, long deviceID);
Description	<p>This method is invoked if you have reported the event <code>FEUSB_DISCONNECT_EVENT</code> using the method <code>AddEventHandler</code> of the Reader class <code>FedmlscReader</code> and a USB Reader is disconnected from the PC.</p> <p>This procedure is very useful for telling an application that the USB Reader is available.</p> <p>In the first pass parameter a device handle is passed. This has meaning only if you are administering multiple USB Readers using the class <code>FeUsb</code>.</p> <p>The second pass parameter contains the serial number of the USB Reader.</p>
Return Value	None
Exceptions	None
Example	see equivalent example in 7.1.60. AddEventListener

7.8.FedmTaskListener

7.8.1. OnNewTag

Function	Method signals new tag(s) in ISO-Table
Syntax	void OnNewTag(int error);
Description	<p>This method is invoked after the reader has sent an inventory notification and the transponder data are collected in the ISO-Table.</p> <p>The access to the transponder data is exactly as after sent of a [0xB0][0x01] Inventory. For more informations, see 8.2.2.Examples for using the ISO table with [0xB0] Commands.</p> <p>The reception of notifications must be enabled with the method StartAsyncTask.</p>
Cross-reference	7.1.62. StartAsyncTask
Exceptions	None

7.8.2. OnNewNotification

Function	Method signals new tag(s) in BRM-Table
Syntax	void OnNewNotification(int error, string ip, uint portNr);
Description	<p>This method is invoked after the reader has sent an inventory notification and the transponder data are collected in the BRM-Table.</p> <p>The access to the transponder data is exactly as after sent of a [0x22] Read Buffer command. For more informations, see 8.2.4. Commands for Buffered Read Mode.</p> <p>The parameter <i>ip</i> is the IP-Address of the reader and <i>portNr</i> is the local port number, which has received the notification.</p> <p>The reception of notifications must be enabled with the method StartAsyncTask.</p>
Cross-reference	7.1.62. StartAsyncTask
Exceptions	None

7.8.3. OnNewReaderDiagnostic

Function	Method signals new reader diagnostic values
Syntax	void OnNewReaderDiagnostic(int error, string ip, uint portNr);
Description	<p>This method is invoked after the reader has sent diagnostic data and the values are saved in the data container TmpData.</p> <p>The access to the diagnostic data is as follows:</p> <pre>byte[] data; GetData(FEDM_ISC_TMP_DIAG_DATA, out data);</pre> <p>For the interpretation of the diagnostic data, please refer to the system manual of the reader. The diagnostic data sent with a notification are generated with mode 0x01.</p> <p>The parameter <i>ip</i> is the IP-Address of the reader and <i>portNr</i> is the local port number, which has received the notification.</p> <p>The reception of notifications must be enabled with the method StartAsyncTask.</p>
Cross-reference	7.1.62. StartAsyncTask
Exceptions	None

7.8.4. OnNewPeopleCounterEvent

Function	Method signals new counter values from a Gate People Counter
Syntax	void OnNewPeopleCounterEvent(uint counter1, uint counter2, uint counter3, uint counter4, string ip, uint portNr, uint busAddress);
Description	<p>This method is invoked after the Gate People Counter has incremented one ore more counters and the Reader has sent this notification.</p> <p>The parameters <i>counter1..4</i> contain the counter values.</p> <p>The parameter <i>ip</i> is the IP-Address of the reader and <i>portNr</i> is the local port number, which has received the notification.</p> <p>The parameter <i>busAddress</i> is the bus address of the Gate People Counter.</p> <p>The reception of notifications must be enabled with the method StartAsyncTask.</p>
Cross-reference	7.1.62. StartAsyncTask
Exceptions	None

7.8.5. OnNewSAMResponse

Function	Method signals new response data from SAM
Syntax	void OnNewSAMResponse(int error, byte[] responseData);
Description	This method is invoked after the SAM inside the Reader has sent the response data (error = 0).
Cross-reference	7.1.23. SendSAMCommand
Exceptions	None

7.8.6. OnNewApduResponse

Function	Method signals new APDU response data from transponder
Syntax	void OnNewApduResponse(int error);
Description	<p>This method is invoked after the transponder has sent the APDU response data (error = 0).</p> <p>The value of error can be:</p> <ul style="list-style-type: none">- 0 : APDU response data is valid- <0: error code and Apdu response is not valid- >0: status value from last responded protocol. Especially 0x96 means an ISO 14443 error and the additional error code can be requested with GetTclIsoErrorCode.
Cross-reference	7.1.21. SendTclApdu , SendTclPing , SendTclDeselect
Exceptions	None

7.8.7. OnNewQueueResponse

Function	Method signals new response data after a Command Queue operation.
Syntax	void OnNewQueueResponse(int error);
Description	This method is invoked after the Command Queue operation is finished and the Reader has sent the response data (error = 0).
Cross-reference	7.1.22. SendCommandQueue
Exceptions	None

7.9.FedmException

Function	Exception class for FedmlscReader
Description	In case of error a new instance of this class is thrown.

7.10.FedmPortDriverException

Function	Exception class for FedmlscReader.
Description	In case of error a new instance of this class is thrown.

7.11.FedmReaderDriverException

Function	Exception class for FedmlscReader.
Description	In case of error a new instance of this class is thrown.

8.Examples for using the function `SendProtocol`

The function *SendProtocol* of the reader class and function unit class is vitally important for the protocol transfer. For this reason an example is shown for each control byte⁶, which is intended to clarify which data are to be saved in data containers with which access constants before each protocol transfer, and which data are available after the protocol transfer. Some protocols allow various data to be transferred. In such a case only a typical example is shown.

All access constants are contained in the structure **FedmlscReaderID** respectively **FedmlscFunctionUnitID** and should be studied thoroughly together with the explanation of protocol data contained in the system manual for the Reader.

For reasons of clarity, the processes for evaluating return values and catching exceptions are omitted here. These processes should however always be performed in applications. Especially for the method `SendProtocol(..)` the evaluation of the return value is mandatory.

In the examples below it is assumed that the reader class **FedmlscReader** and the structures **FedmlscReaderID** and **FedmlscReaderConst** as well as **FedmlscFunctionUnit** and **FedmlscFunctionUnitID** are incorporated:

```
using OBID.FedmlscReader;  
using OBID.FedmlscReaderID;  
using OBID.FedmlscReaderConst;  
  
using OBID.FedmlscFunctionUnit;  
using OBID.FedmlscFunctionUnitID;
```

The reader object shall be defined as:

```
FedmlscReader reader = new FedmlscReader;
```

The function unit shall be defined as:

```
FedmlscFunctionUnit fu = new FedmlscFunctionUnit;
```

⁶not all commands are supported by every Reader. Detailed informations about the supported commands can be found in the system manual of the Reader.

8.1. Basic commands

[Control Byte] Protocol	Example ⁷
[0x18] Destroy	<pre> byte mode = 0; // Mode (always 0) byte epcLen = 0; // Number of bytes in EPC string epc; // EPC string pw; // Password // take the data e.g from an input field // get the length of EPC epcLen = epc.Length reader.SetData(FEDM_ISC_TMP_EPC_DESTROY_MODE, (byte)0); reader.SetData(FEDM_ISC_TMP_EPC_DESTROY_LEN, epcLen); reader.SetData(FEDM_ISC_TMP_EPC_DESTROY_PASSWORD, pw); reader.SetData(FEDM_ISC_TMP_EPC_DESTROY_EPC, epc); reader.SendProtocol(0x18); </pre>
[0x1A] Halt	<pre> reader.SendProtocol(0x1A); </pre>
[0x1B] Reset QUIET Bit	<pre> reader.SendProtocol(0x1B); </pre>
[0x1C] EAS	<pre> reader.SendProtocol(0x1C); </pre>
[0x21]Read Buffer (only ID ISC.LR200 and ID ISC.LR2000)	<pre> byte dataSets = 1; // Number requested data sets byte trData = 0; // Data set structure byte recSets = 0; // Number of data sets in receive protocol reader.SetData(FEDM_ISCLR_TMP_BRM_SETS, dataSets); reader.SendProtocol(0x21); // read data blocks from transponder using Buffered Read Mode reader.GetData(FEDM_ISCLR_TMP_BRM_TRDATA, out trData); reader.GetData(FEDM_ISCLR_TMP_BRM_RECSETS, out recSets); // All other transponder data are enclosed in the BRMTable. Examples for dataaccess in // 8.2.4. Commands for Buffered Read Mode </pre>
[0x22]Read Buffer (for all readers with Buffered Read Mode, except for ID ISC.LR200)	<pre> uint dataSets = 1; // Number requested data sets byte trData = 0; // Data set structure uint recSets = 0; // Number of data sets in receive protocol reader.SetData(OBID.ReaderCommand._0x22.Req.DATA_SETS, dataSets); reader.SendProtocol(0x22); // read data blocks from transponder using Buffered Read Mode reader.GetData(OBID.ReaderCommand._0x22.Rsp.TR_DATA1, out trData); reader.GetData(OBID.ReaderCommand._0x22.Rsp.DATA_SETS, out recSets); // All other transponder data are enclosed in the BRMTable. Examples for dataaccess in // 8.2.4. Commands for Buffered Read Mode </pre>

⁷ all examples in C#

[Control Byte] Protocol	Example ⁷
[0x31] Read Data Buffer Info	<pre>uint tabSize = 0; // Size of data buffer uint tabStart = 0; // Start address of the first data set uint tabLen = 0; // Number of data sets in the data buffer reader.SendProtocol(0x31); reader.GetData(OBID.ReaderCommand._0x31.Rsp.TAB_SIZE, out tabSize); reader.GetData(OBID.ReaderCommand._0x31.Rsp.TAB_START, out tabStart); reader.GetData(OBID.ReaderCommand._0x31.Rsp.TAB_LEN, out tabLen);</pre>
[0x32] Clear Data Buffer	<pre>reader.SendProtocol(0x32);</pre>
[0x33] Initialize Buffer	<pre>reader.SendProtocol(0x33);</pre>
[0x34] Force Notify Trigger	<pre>reader.SendProtocol(0x34);</pre>
[0x52] Baud Rate Detection	<pre>reader.SendProtocol(0x52);</pre>
[0x55] Start Flash Loader	<pre>reader.SendProtocol(0x55);</pre>
[0x63] CPU Reset	<pre>reader.SendProtocol(0x63);</pre>
[0x64] System Reset	<pre>byte cMode = 0; // LRU1000 RF-Controller (1 for LRU1000 AC-Controller) reader.SetData(OBID.ReaderCommand._0x64.Req.MODE, cMode); reader.SendProtocol(0x64);</pre>
[0x65] Get Software Version	<pre>string softVer; // Software version as string reader.SendProtocol(0x65); reader.GetData(FEDM_ISC_TMP_SOFTVER, out softVer);</pre>
[0x66] Get Reader Info	<pre>string sInfo; // Reader Info as String reader.SetData(OBID.ReaderCommand._0x66.Req.MODE, (uint)0); // identical with [0x65] //reader.SetData(ReaderCommand._0x66.Req.MODE, (uint)1); // LRU1000: AC-Controller reader.SendProtocol(0x66); reader.GetData(OBID.ReaderCommand._0x66.Rsp.READER_INFO, sInfo);</pre>
[0x69] RF Reset	<pre>reader.SendProtocol(0x69);</pre>
[0x6A] RF ON/OFF	<pre>byte RF = 1; // RFON reader.SetData(OBID.ReaderCommand._0x6A.Req.RF_OUTPUT, RF); reader.SendProtocol(0x6A);</pre>
[0x6C] Set Noise Level	<pre>uint NLMin = 500; // minimum noise level uint NLAvg = 1000; // average noise level uint NLMax = 1500; // maximum noise level reader.SetData(FEDM_ISC_TMP_NOISE_LEVEL_MIN, NLMin); reader.SetData(FEDM_ISC_TMP_NOISE_LEVEL_AVG, NLAvg); reader.SetData(FEDM_ISC_TMP_NOISE_LEVEL_MAX, NLMax); reader.SendProtocol(0x6C);</pre>

[Control Byte] Protocol	Example ⁷
[0x6D] Get Noise Level	<pre> uint NLMin = 0; // minimum noise level uint NLAvg = 0; // average noise level uint NLMax = 0; // maximum noise level reader.SendProtocol(0x6D); reader.GetData(OBID.ReaderCommand._0x6D.Rsp.NL_MIN, out NLMin); reader.GetData(OBID.ReaderCommand._0x6D.Rsp.NL_AVG, out NLAvg); reader.GetData(OBID.ReaderCommand._0x6D.Rsp.NL_MAX, out NLMax); </pre>
[0x6E] Reader Diagnostic	<pre> byte diagMode = 1; // Diagnostic mode string sData; // Diagnostic Data as string reader.SetData(OBID.ReaderCommand._0x6E.Req.MODE, diagMode); reader.SendProtocol(0x6E); reader.GetData(OBID.ReaderCommand._0x6E.Rsp.DIAGNOSTIC_DATA, sData); </pre>
[0x6F] Base Antenna Tuning	<pre> reader.SendProtocol(0x6F); // The Long-Range-Reader changes into the spezial mode // The mode can be left only by performing a reset </pre>
[0x71] Set Output	<pre> // Example 1 from the system manual ID ISC.M01 reader.SetData(OBID.ReaderCommand._0x71.Req.OUTPUT_STATE, 0); // OS-Bytes reset reader.SetData(OBID.ReaderCommand._0x71.Req.OutputState.OUT1, (byte)0x01); // Output 1 active reader.SetData(OBID.ReaderCommand._0x71.Req.OutputState.LED_GREEN, (byte)0x10); // LED green off reader.SetData(OBID.ReaderCommand._0x71.Req.OutputState.LED_RED, (byte)0x01); // LED red on reader.SetData(OBID.ReaderCommand._0x71.Req.OutputState.BEEPER, (byte)0x11); // Beeper alternated on reader.SetData(OBID.ReaderCommand._0x71.Req.OUTPUT_STATE_FLASH, (int)0); //OSF-Bytes reset reader.SetData(OBID.ReaderCommand._0x71.Req.OutputStateFlash.BEEPER, (byte)0x01); // Beeper with 4Hz reader.SetData(OBID.ReaderCommand._0x71.Req.OS_TIME, (uint)5); // 500ms active time Beeper and LED's reader.SetData(OBID.ReaderCommand._0x71.Req.OUT_TIME, (uint)3); // Output 1 300ms active reader.SendProtocol(0x71); </pre>

[Control Byte] Protocol	Example ⁷
[0x72] Set Output	<pre>// Example from the system manual ID ISC.LRU1000 reader.SetData(OBID.ReaderCommand._0x72.Req.MODE, (byte)0x00); // set mode to 0 reader.SetData(OBID.ReaderCommand._0x72.Req.OUT_N, (byte)0x03); // activate 3 outputs reader.SetData(OBID.ReaderCommand._0x72.Req.No1.OUT_NUMBER, (byte)0x01); // output 1 reader.SetData(OBID.ReaderCommand._0x72.Req.No1.OUT_TYPE, (byte)0x00);// type: general output reader.SetData(OBID.ReaderCommand._0x72.Req.No1.State.MODE, (byte)0x03); // alternating reader.SetData(OBID.ReaderCommand._0x72.Req.No1.State.FREQUENCY, (byte)0x01); // 4 Hz reader.SetData(OBID.ReaderCommand._0x72.Req.No1.OUT_TIME, (uint)5); // 500 ms reader.SetData(OBID.ReaderCommand._0x72.Req.No2.OUT_NUMBER, (byte)0x01); // relais 1 reader.SetData(OBID.ReaderCommand._0x72.Req.No2.OUT_TYPE, (byte)0x04); // type: relais reader.SetData(OBID.ReaderCommand._0x72.Req.No2.State.MODE, (byte)0x02); // switching off reader.SetData(OBID.ReaderCommand._0x72.Req.No2.State.FREQUENCY, (byte)0x00);// unchanged reader.SetData(OBID.ReaderCommand._0x72.Req.No2.OUT_TIME, (uint)2); // 200 ms reader.SetData(OBID.ReaderCommand._0x72.Req.No3.OUT_NUMBER, (byte)0x02); // relais 2 reader.SetData(OBID.ReaderCommand._0x72.Req.No3.OUT_TYPE, (byte)0x04); // type: relais reader.SetData(OBID.ReaderCommand._0x72.Req.No3.State.MODE, (byte)0x01); // switching on reader.SetData(OBID.ReaderCommand._0x72.Req.No3.State.FREQUENCY, (byte)0x00);// unchanged reader.SetData(OBID.ReaderCommand._0x72.Req.No3.OUT_TIME, (uint)10); // 1000 ms reader.SendProtocol((0x72);</pre>
[0x74] Get Input	<pre>// Example for ID ISC.LR2500-B bool in1 = false; // Input 1 bool in2 = false; // Input 2 bool in3 = false; // Input 3 reader.SendProtocol(0x74); reader.GetData(OBID.ReaderCommand._0x74.Rsp.Inputs.IN1, out in1); reader.GetData(OBID.ReaderCommand._0x74.Rsp.Inputs.IN2, out in2); reader.GetData(OBID.ReaderCommand._0x74.Rsp.Inputs.IN3, out in3);</pre>
[0x75] Adjust Antenna	<pre>int antValue = 0; // Antenna voltage reader.SendProtocol(0x75); reader.GetData(FEDM_ISCM_TMP_ ANTENNA_VALUE, out antValue);</pre>

[Control Byte] Protocol	Example ⁷
<p>[0x80] Read Configuration</p> <p>and</p> <p>[0x81] Write Configuration</p>	<pre>// The sample shows the read and write back function of one block in the reader configuration byte cfgAdr = 2; // Adress of the configuration block bool eeProm = true; // Configuration data from/into EEPROM of reader byte busAddress; // Bus address of the ISC.LR-Lesers from Block 2 reader.SetData(OBID.ReaderCommand._0x80.Req.CFG_ADDRESS, (byte)0x00); //reset all reader.SetData(OBID.ReaderCommand._0x80.Req.CfgAddr.ADDRESS, cfgAdr); // set address reader.SetData(OBID.ReaderCommand._0x80.Req.CfgAddr.LOCATION, eeProm); // memory location on EEPROM reader.SendProtocol(0x80); // write back configuration data //take over bus address reader.GetConfigPara(OBID.ReaderConfig.HostInterface.Serial.BusAddress, out busAddress); reader.SetData(OBID.ReaderCommand._0x81.Req.CFG_ADDRESS, (byte)0x00); //reset all reader.SetData(OBID.ReaderCommand._0x81.Req.CfgAddr.ADDRESS, cfgAdr); //set address reader.SetData(OBID.ReaderCommand._0x81.Req.CfgAddr.LOCATION, eeProm); // memory location on EEPROM reader.SendProtocol(0x81); // write back configuration data</pre>
<p>[0x83] Set Default Configuration</p>	<pre>reader.SetData(OBID.ReaderCommand._0x83.Req.CFG_ADDRESS, (byte)0x00); // reset all reader.SetData(OBID.ReaderCommand._0x83.Req.CfgAddr.ADDRESS, (byte)0x02); //set address reader.SetData(OBID.ReaderCommand._0x83.Req.CfgAddr.LOCATION, false); // choose RAM reader.SetData(OBID.ReaderCommand._0x83.Req.CfgAddr.MODE, false); // set default only block 2 reader.SendProtocol(0x83); // Set configuration data from block 2 in RAM to default</pre>
<p>[0x85] Set System Timer</p>	<pre>reader.SetData(OBID.ReaderCommand._0x85.Req.TIMER_HOUR, (uint)16); // 16 hours reader.SetData(OBID.ReaderCommand._0x85.Req.TIMER_MINUTE, (uint)20); // 20 minutes reader.SetData(OBID.ReaderCommand._0x85.Req.TIMER_MILLISECONDS, (uint)2000); // 2000 milliseconds reader.SendProtocol(0x85); //set Timer</pre>
<p>[0x86] Get System Timer</p>	<pre>uint hour = 0; // hours uint minute = 0; // minutes uint milliSec = 0; // milliseconds reader.SendProtocol(0x86); // read timer reader.GetData(OBID.ReaderCommand._0x86.Req.TIMER_HOUR, out hour); // take over hours reader.GetData(OBID.ReaderCommand._0x86.Req.TIMER_MINUTE, out minute); // take over minutes reader.GetData(OBID.ReaderCommand._0x86.Req.TIMER_MILLISECONDS, out milliSec); //take over milliseconds</pre>

[Control Byte] Protocol	Example ⁷
[0x87] Set System Date	<pre> reader.SetData(OBID.ReaderCommand._0x87.Req.DATE_CENTURY, (uint)20); // 20. century reader.SetData(OBID.ReaderCommand._0x87.Req.DATE_YEAR, (uint)4); // year 04 in the century reader.SetData(OBID.ReaderCommand._0x87.Req.DATE_MONTH, (uint)9); // September reader.SetData(OBID.ReaderCommand._0x87.Req.DATE_DAY, (uint)15); // 15. September reader.SetData(OBID.ReaderCommand._0x87.Req.DATE_TIMEZONE, (uint)0); // actually unused reader.SetData(OBID.ReaderCommand._0x87.Req.TIME_HOUR, (uint)12); // hours reader.SetData(OBID.ReaderCommand._0x87.Req.TIME_MINUTE, (uint)00); // minutes reader.SetData(OBID.ReaderCommand._0x87.Req.TIME_MILLISECOND, (uint)0); // milliseconds (incl. seconds) reader.SendProtocol(0x87); // set date and time </pre>
[0x88] Get System Date	<pre> byte cCentury = 0; // century byte cYear = 0; // year in the century byte cMonth = 0; // month byte cDay = 0; // day byte cTimezone = 0; // timezone (actually unused) byte cHour = 0; // hours byte cMinute = 0; // minutes uint uiMilliSec = 0; // milliseconds reader.SendProtocol(0x88); // read date and time reader.GetData(OBID.ReaderCommand._0x88.Req.DATE_CENTURY, out cCentury); // century reader.GetData(OBID.ReaderCommand._0x88.Req.DATE_YEAR, out cYear); // year in the century reader.GetData(OBID.ReaderCommand._0x88.Req.DATE_MONTH, out cMonth); // month reader.GetData(OBID.ReaderCommand._0x88.Req.DATE_DAY, out cDay); // day reader.GetData(OBID.ReaderCommand._0x88.Req.DATE_TIMEZONE, out cTimezone); // actually unused reader.GetData(OBID.ReaderCommand._0x88.Req.TIME_HOUR, out cHour); // hours reader.GetData(OBID.ReaderCommand._0x88.Req.TIME_MINUTE, out cMinute); // minuts reader.GetData(OBID.ReaderCommand._0x88.Req.TIME_MILLISECOND, out uiMilliSec); // milliseconds </pre>
[0x8A] Read Configuration und [0x8B] Write Configuration	<pre> // the example shows reading and resetting a reader configuration block byte CfgAdr = 2; // address of the configuration block byte BusAddress; // bus address of ISC.LRU3000 from Block 1 reader.SetData(OBID.ReaderCommand._0x8A.Req.DEVICE, (byte)0x02); // RF-Controller reader.SetData(OBID.ReaderCommand._0x8A.Req.BANK, (byte)0x01); // bank Main reader.SetData(OBID.ReaderCommand._0x8A.Req.MODE, (byte)0x00); // clear mode byte reader.SetData(OBID.ReaderCommand._0x8A.Req.Mode.LOCATION, true); // EEPROM reader.SetData(OBID.ReaderCommand._0x8A.Req.CFG_ADDRESS, CfgAdr); // configuration address reader.SetData(OBID.ReaderCommand._0x8A.Req.CFG_N, (byte)1); // 1 configuration block reader.SendProtocol(0x8A); // execute command // retrieve the busaddress reader.GetConfigPara(OBID.ReaderConfig.HostInterface.Serial.BusAddress, out BusAdr); </pre>

[Control Byte] Protocol	Example ⁷
	<pre>// change parameters with reader.SetConfigPara(ReaderConfig.,); reader.SetData(OBID.ReaderCommand._0x8B.Req.DEVICE, (byte)0x02); // RF-Controller reader.SetData(OBID.ReaderCommand._0x8B.Req.BANK, (byte)0x01); // bank Main reader.SetData(OBID.ReaderCommand._0x8B.Req.MODE, (reader.)0x00); // clear byte reader.SetData(OBID.ReaderCommand._0x8B.Req.Mode.LOCATION, true); // EEPROM reader.SetData(OBID.ReaderCommand._0x8B.Req.CFG_ADDRESS, CfgAdr); // configuration address reader.SetData(OBID.ReaderCommand._0x8B.Req.CFG_N, (byte)1); // 1 configuration block reader.SendProtocol(0x8B); // execute command</pre>

[Control Byte] Protocol	Example ⁷
[0x8C] Set Default Configuration	<pre> reader.SetData(OBID.ReaderCommand._0x8C.Req.DEVICE, (byte)0x02); // RF-Controller reader.SetData(OBID.ReaderCommand._0x8C.Req.BANK, (byte)0x01); // bank Main reader.SetData(OBID.ReaderCommand._0x8C.Req.MODE, (byte)0x00); // clear byte reader.SetData(OBID.ReaderCommand._0x8C.Req.Mode.LOCATION, true); // EEPROM reader.SetData(OBID.ReaderCommand._0x8C.Req.CFG_ADDRESS, (byte)1); //configuration address reader.SetData(OBID.ReaderCommand._0x8C.Req.CFG_N, (byte)1); // 1 configuration block reader.SendProtocol(0x8C); // execute command </pre>
[0xA0] Reader Login	<pre> string passWord; // reader password // take the password e.g. from an input field reader.SetData(OBID.ReaderCommand._0xA0.Req.PASSWORD, passWord); //set password reader.SendProtocol(0xA0); // send password to reader </pre>
[0xA2] Write Mifare Keys	<pre> string key; // Mifare-Key // take the Mifare key e.g. from an input field reader.SetData(OBID.ReaderCommand._0xA2.Req.KEY_TYPE, (byte)0); reader.SetData(OBID.ReaderCommand._0xA2.Req.KEY_ADR, (byte)0); reader.SetData(OBID.ReaderCommand._0xA2.Req.KEY, key); reader.SendProtocol(0xA2); // send Mifare-Key to the reader </pre>
[0xA3] Write AES/DES Keys	<pre> string key; // Key // take the key e.g. from an input field reader.SetData(OBID.ReaderCommand._0xA3.Req.MODE, (byte)0); // RAM reader.SetData(OBID.ReaderCommand._0xA3.Req.KEY_INDEX, (byte)0); reader.SetData(OBID.ReaderCommand._0xA3.Req.AUTHENTICATION_MODE,(byte)0); //DESFire native TDES reader.SetData(OBID.ReaderCommand._0xA3.Req.KEY_LEN, key.length); reader.SetData(OBID.ReaderCommand._0xA3.Req.KEY, key); reader.SendProtocol(0xA3); // send key to the Reader </pre>
[0xAD] Write Reader Authent Key	<pre> string key; // Authent-Key // take the key e.g. from an input field reader.SetData(OBID.ReaderCommand._0xAD.Req.KEY_TYPE, (byte)2); // AES256 reader.SetData(OBID.ReaderCommand._0xAD.Req.KEY_LEN, (byte)32); reader.SetData(OBID.ReaderCommand._0xAD.Req.KEY, key); reader.SendProtocol(0xAD); // write Authent-Key into Reader </pre>

[Control Byte] Protocol	Example ⁷
[0xB0] ISO Mandatory and Optional Commands	<pre>// the sample shows the [0x01] Inventory reader.SetData(OBID.ReaderCommand._0xB0.SUB_COMMAND, (byte)0x01); // Inventory reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x01.Req.MODE, (byte)0x00); // no More-Flag reader.SendProtocol(0xB0); // the Inventory-data are in the ISO-Table object. Sample for data access in 8.2.2.Examples for using the ISO table with [0xB0] Commands</pre>
[0xB1] ISO15693 Customer and Proprietary Commands (TagHandler-Classes provide an easier API)	<pre>// The Sample shows the [0xA2] Set EAS // all others corresponds to the 0xB1-Commands string snr = new string(); // for Serialnumber byte isoError = 0; // for ISO-errorcode reader.SetData(FEDM_ISC_TMP_B1_CMD, (byte)0xA2); // Set EAS reader.SetData(FEDM_ISC_TMP_B1_MFR, (byte) ISO_MFR_PHILIPS); //Manu. reader.SetData(FEDM_ISC_TMP_B1_MODE, (byte) ISO_MODE_ADR); // addr. // ... Serial number e.g. take from text field and store in snr reader.SetData(FEDM_ISC_TMP_B1_REQ_UID, snr); int status = reader.SendProtocol(0xB1); if(status == 0x95) { // take ISO-Error code reader.GetData(FEDM_ISC_TMP_B1_ISO_ERROR, out isoError); }</pre>
[0xB2] ISO14443 Special Commands [0x2B] ISO14443-4 Transponder Info (TagHandler-Classes provide an easier API)	<pre>byte cFSCI = 0; byte cFWI = 0; byte cDSI = 0; byte cDRI = 0; byte cNad = 0; byte cCid = 0; reader.SetData(FEDM_ISC_TMP_B2_CMD, (byte)0x2B); // ISO14443-4 Transponder Info int iStatus = reader.SendProtocol(0xB2); // transponder must previously selected with // [0x25] Select if(iStatus == 0x00) { // get the table index of the selected transponder int ildx = reader.FindTableIndex(0, ISO_TABLE, DATA_IS_SELECTED, true); if(ildx >= 0) { // get transponder data reader.GetTableData(ildx, ISO_TABLE, DATA_FSCI, out cFSCI) reader.GetTableData(ildx, ISO_TABLE, DATA_FWI, out cFWI) reader.GetTableData(ildx, ISO_TABLE, DATA_DSI, out cDSI) reader.GetTableData(ildx, ISO_TABLE, DATA_DRI, out cDRI) reader.GetTableData(ildx, ISO_TABLE, DATA_NAD, out cNad) reader.GetTableData(ildx, ISO_TABLE, DATA_CID, out cCid) } }</pre>

[Control Byte] Protocol	Example ⁷
<p>[0xB2] ISO14443 Special Commands</p> <p>[0xB0] Authent Mifare</p> <p>(TagHandler-Classes provide an easier API)</p>	<pre> byte dbAddress = 0; // Address of data block byte keyType = 0; // Key type for authentication byte keyAdr = 0; // EEPROM-address of the keys in the reader byte keyLocation = 0; // Location of the Authent-Key (0: Reader; 1: Protocol) string key = "000000000000"; // Authent-Key reader.SetData(FEDM_ISC_TMP_B2_CMD, (byte)0xB0); // Authent Mifare reader.SetData(FEDM_ISC_TMP_B2_MODE, (byte)0x00); // clear mode byte reader.SetData(FEDM_ISC_TMP_B2_MODE, (byte)FEDM_ISC_ISO_MODE_SEL); //selected reader.SetData(FEDM_ISC_TMP_B2_REQ_KEY_TYPE, keyType); reader.SetData(FEDM_ISC_TMP_B2_REQ_DB_ADR, dbAddress); reader.SetData(FEDM_ISC_TMP_B2_MODE_KL, keyLocation); if(keyLocation == 0) reader.SetData(FEDM_ISC_TMP_B2_REQ_KEY_ADR, keyAdr); else reader.SetData(FEDM_ISC_TMP_ISO14443A_KEY, key); reader.SendProtocol(0xB2); </pre>
<p>[0xB2] ISO14443 Special Commands</p> <p>[0xB1] Authent my-d</p> <p>(TagHandler-Classes provide an easier API)</p>	<pre> byte keyAdrTag = 5; // Address of the keys on the transponder byte keyAdrSam = 2; // Address of the keys in the authentication module byte cntAdr = 3; // Address of the authentication counter byte authSeq = 0; // Authentication sequence reader.SetData(FEDM_ISC_TMP_B2_CMD, (byte)0xB1); // Authent my-d reader.SetData(FEDM_ISC_TMP_B2_MODE, (byte)ISO_MODE_SEL); // selected reader.SetData(FEDM_ISC_TMP_B2_REQ_KEY_ADR_TAG, keyAdrTag); reader.SetData(FEDM_ISC_TMP_B2_REQ_KEY_ADR_SAM, keyAdrSam); reader.SetData(FEDM_ISC_TMP_B2_REQ_AUTH_COUNTER_ADR, cntAdr); reader.SetData(FEDM_ISC_TMP_B2_REQ_KEY_AUTH_SEQUENCE, authSeq); reader.SendProtocol(0xB2); </pre>
<p>[0xB2] ISO14443 Special Commands</p> <p>[0xB2] Authent Mifare Ultralight C</p> <p>(TagHandler-Classes provide an easier API)</p>	<pre> byte keyIndex = 0; // reader key index for authentication reader.SetData(FEDM_ISC_TMP_B2_CMD, (byte)0xB2); // Authent Mifare Ultralight C reader.SetData(FEDM_ISC_TMP_B2_MODE, (byte)0x00); // clear mode byte reader.SetData(FEDM_ISC_TMP_B2_MODE, (byte)FEDM_ISC_ISO_MODE_SEL); //selected reader.SetData(FEDM_ISC_TMP_B2_REQ_KEY_INDEX, keyIndex); reader.SendProtocol(0xB2); </pre>

[Control Byte] Protocol	Example ⁷
[0xB2] ISO14443 Special Commands [0x30] Mifare Value Commands (TagHandler-Classes provide an easier API)	<pre> byte mfCmd = 0x01; // Mifare Command byte dbAdr = 0x05; // datablock address byte[] opValue = new byte[4]; // OP_VALUE byte destAdr = 0x05; // destination address opValue[0] = 0x00; opValue[1] = 0x00; opValue[2] = 0x00; opValue[3] = 0x03; reader.SetData(FEDM_ISC_TMP_B2_CMD, (byte)0x30); // Mifare Value Commands reader.SetData(FEDM_ISC_TMP_B2_MODE, (byte) FEDM_ISC_ISO_MODE_SEL); // selected reader.SetData(FEDM_ISC_TMP_B2_REQ_MF_CMD, mfCmd); reader.SetData(FEDM_ISC_TMP_B2_REQ_DB_ADR, dbAdr); reader.SetData(FEDM_ISC_TMP_B2_REQ_OP_VALUE, opValue); reader.SetData(FEDM_ISC_TMP_B2_REQ_DEST_ADR, destAdr); reader.SendProtocol(0xB2); </pre>

8.2. Table oriented commands

Table oriented commands can only be used when the specified table (table for Host-Commands or table for Buffered Read Mode) is dimensioned prior ([5.1.1. Initializing](#)).

If Transponders with more than 256 data blocks are in use together with Host-Commands, the ISO table must be prepared with the method `SetTableSize (2)` in [7.1.52. SetTableSize](#).

8.2.1. Anomaly of the addressed mode

Most of the Host Commands can be used in the addressed mode. In this case the serial number – or unified identifier (UID) – is part of the send protocol. In former versions the library has only supported UIDs with a length of 8 byte. With an extension flag in the mode byte (UID_LF) different UID length are now possible. If the UID_LF flag is set, the length of the UID must be added to the send protocol.

The following example demonstrates the use of a different UID length in a [0xB0][0xB23] Read Multiple Blocks:

```
// set UID for addressed mode (up to 96 byte)
SetData(ReaderCommand._0xB0.SubCmd._0x23.Req.UID, sUid);
SetData(ReaderCommand._0xB0.SubCmd._0x23.Req.ExtAddMode.UID_LEN, cUidLen);    // number of byte in UID

SetData(ReaderCommand._0xB0.SUB_COMMAND, (byte)0x23);                      // Command Read Multiple Blocks
SetData(ReaderCommand._0xB0.SubCmd._0x23.Req.MODE, (byte)0x00);             // clear mode byte
SetData(ReaderCommand._0xB0.SubCmd._0x23.Req.Mode.ADR, (byte)0x01);         // addressed mode
SetData(ReaderCommand._0xB0.SubCmd._0x23.Req.Mode.UID_LF, true);            // UID_LF flag
SetData(ReaderCommand._0xB0.SubCmd._0x23.Req.ExtAddrMode.DBN, (byte)0x01);  // request one data block
SetData(ReaderCommand._0xB0.SubCmd._0x23.Req.ExtAddrMode.DB_ADR, ucDBAdr);  // set data block address

SendProtocol(0xB0); // communication wit reader/transponder
```

8.2.2.Examples for using the ISO table with [0xB0] Commands

[Control byte] protocol	Example ⁸
[0x01] Inventory for HF-Transponder: - Philips I-CODE1 - Texas Instruments Tag-it HF - ISO15693 - ISO14443A - ISO14443B - EPC (Electronic Product Code) - Philips I-CODE UID - Innovision Jewel - ISO 18000-3M3 for UHF-Transponder: - ISO18006-6-B	<pre>byte trType = 0; // for transponder type string snr; // for serial number (also EPC) string header; // for EPC header string domain; // for EPC domain manager field string object ; // for EPC object class field string epc; // for EPC ("Header.DomainManager.ObjectClass.SerialNumber") reader.SetData(OBID.ReaderCommand._0xB0.SUB_COMMAND, (byte)0x01); // Command Inventory reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x01.Req.MODE, (byte)0x00); // no more flag // set table length to 0 and delete the content of the table completely reader.ResetTable(ISO_TABLE); reader.SendProtocol(0xB0); // Communication with reader/transponder</pre>

⁸ all examples in C#

[Control byte] protocol	Example ⁸
<ul style="list-style-type: none"> - EM4222 - EPC Class0/0+ - EPC Class1 Gen1 - EPC Class1 Gen2 	<pre> // All transponder data are in the Table for(int cnt=0; cnt< reader.GetTableLength(FedmlscReaderConst.ISO_TABLE); ++cnt) { // take transponder type reader.GetTableData(cnt,ISO_TABLE, DATA_TRTYPE, out trType); switch(trType) { case 0x00: // Philips I-CODE1 case 0x01: // Texas Instruments Tag-it HF case 0x03: // ISO15693 case 0x04: // ISO14443A case 0x05: // ISO14443B case 0x07: // I-Code UID case 0x08: // Innovision Jewel case 0x09: // ISO 18000-3M3 case 0x81: // ISO18000-6-B case 0x83: // EM4222 case 0x84: // EPC Class1 Gen2 case 0x88: // EPC Class0/0+ case 0x89: // EPC Class1 Gen1 // take serial number as string reader.GetTableData(cnt, ISO_TABLE, DATA_SNR, out snr); break; case 0x06: // EPC (Electronic Product Code) // take EPC-Fields reader.GetTableData(cnt, ISO_TABLE, DATA_EPC_HEADER, out header); reader.GetTableData(cnt, ISO_TABLE, DATA_EPC_DOMAIN, out domain); reader.GetTableData (cnt, ISO_TABLE, DATA_EPC_OBJECT, out object); reader.GetTableData (cnt, ISO_TABLE, DATA_EPC_SNR, out snr); // or take EPC-Field as complete string reader.GetTableData (cnt, ISO_TABLE, DATA_EPC, out epc); break; } } </pre>
[0x02] Stay Quiet	<pre> string snr; // for serial number // ... take serial number e. g. from textfield and store it in snr // set serial number for addressed mode reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x02.Req.UID, snr); reader.SetData(OBID.ReaderCommand._0xB0.SUB_COMMAND, (byte)0x02);//Command Stay Quiet reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x02.Req.MODE, (byte)0x00);//Mode-Byte zurücks. reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x02.Req.Mode.ADR, (byte)0x01);//Addr. Mode reader.SendProtocol(0xB0); // Communication with reader/transponder </pre>
[0x22] Lock Multiple Blocks	<pre> // Attention: with this ISO Command all data blocks will be locked irretrievably! string snr; // for serial number // ... take serial number e. g. from textfield and store it in snr // determine table index of the serial number int idx = reader.FindTableIndex(0,ISO_TABLE, DATA_SNR, snr); if(idx < 0) return; // set serial number for addressed mode </pre>

[Control byte] protocol	Example ⁸
	<pre> reader.SetData(FEDM_ISC_TMP_B0_REQ_UID, snr); reader.SetData(OBID.ReaderCommand._0xB0.SUB_COMMAND, (byte)0x22); // Command Lock Multiple Blocks reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x22.Req.MODE, (byte)0x00); // Mode- Byte reset reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x22.Req.Mode.ADR, (byte)0x01); // Addressed mode reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x22.Req.NormAddrMode.DBN, (byte)0x01); // lock one Data block reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x22.Req.NormAddrMode.DB_ADR, (byte)0x00); // set data block address reader.SendProtocol(0xB0); // Communication with Reader/transponder </pre>
<p>[0x23] Read Multiple Blocks (standard address mode)</p>	<pre> byte[] dataBlock; // buffer for one data block byte dbAddress = 5; // data block address 5 string snr ; // for serial number // ... take serial number e. g. from text field // set serial number for addressed mode reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x23.Req.UID, snr); reader.SetData(OBID.ReaderCommand._0xB0.SUB_COMMAND, (byte)0x23); // Command read multiple blocks reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x23.Req.MODE, (byte)0x00); // Mode byte reset reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x23.Req.Mode.ADR, (byte)0x01); // Addr. Mode eader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x23.Req.NormAddrMode.DBN, (byte)0x01); // read one Data block reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x23.Req.NormAddrMode.DB_ADR, dbAddress); // set Data block address reader.SendProtocol(0xB0); // Communication with reader/transponder // all transponder data are in the table // first determine the table index of the serial number int idx = reader.FindTableIndex(0, ISO_TABLE, DATA_SNR, snr); if(idx < 0) return; // take the size of the data blocks (Block size) byte blockSize; reader.GetTableData(idx, ISO_TABLE, DATA_BLOCKSIZE, out blockSize); // ... do something with the block size // take a data block (data block content only the block size data byte) reader.GetTableData(idx, ISO_TABLE, DATA_RxDB, dbAddress, out dataBlock); // ... do something with the data block </pre>
<p>[0x23] Read Multiple Blocks (extended address mode)</p>	<pre> byte[] dataBlock; // buffer for one data block uint dbAddress = 5; // data block address 5 string snr ; // for serial number string sPw; // for Access Passwort // ... take serial number e. g. from text field // ... take password e. g. from text field // // set serial number (> 8 Byte accpetable) for addressed mode reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x23.Req.UID, snr); reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x23.Req.ExtAddrMode.UID_LEN, snr.Length/2); // length of UID in bytes reader.SetData(OBID.ReaderCommand._0xB0.SUB_COMMAND, (byte)0x23); // Command </pre>

[Control byte] protocol	Example ⁸
	<pre> Read Multiple Blocks reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x23.Req.MODE, (byte)0x00); // clear mode byte reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x23.Req.Mode.ADR, (byte)0x01); // Addr. Mode reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x23.Req.Mode.EXT_ADR, true); // extended addressed mode reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x23.Req.Mode.UID_LF, true); // length of UID != 8 reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x23.Req.ExtAddrMode.BANK, (byte)0x00); // clear bank byte reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x23.Req.ExtAddrMode.Bank.Number, (byte)0x03); // bank User Memory reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x23.Req.ExtAddrMode.Bank.ACCESS_ FLAG, true); // with access password reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x23.Req.ExtAddrMode.ACCESS_PW_L ENGTH, (byte)sPw.Length/2); // len in bytes reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x23.Req.ExtAddrMode.ACCESS_PW, sPw); // password reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x23.Req.ExtAddrMode.DB_ADR, dbAddress); // datablock address reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x23.Req.ExtAddrMode.DBN, (byte)0x01); // read one datablock reader.SendProtocol(0xB0); // Communication with reader/transponder // all transponder data are in the table // first determine the table index of the serial number int idx = reader.FindTableIndex(0, ISO_TABLE, DATA_SNR, snr); if(idx < 0) return; // take the size of the data blocks (Block size) byte blockSize; reader.GetTableData(idx, ISO_TABLE, DATA_BLOCKSIZE, out blockSize); // ... do something with the block size // take a data block (data block contents only the block size data byte) reader.GetTableData(idx, ISO_TABLE, DATA_RxDB, dbAddress, out dataBlock); // ... do something with the data block </pre>
<p>[0x24] Write Multiple Blocks (normal address mode)</p>	<pre> /* The example shows the [0x24] Write Multiple Block. In Addressed Mode an [0x01] Inventory must first be performed. Note: If [0x23] Read Multiple Blocks was not yet carried out, then the block size is preset to 4. But if the transponder in the read field supports another block size, this must first be set in the table for this transponder!! You can use GetTableData(.., DATA_IS_BLOCK_SIZE_SET) to check whether the block size was already read with [0x23] Read Multiple Blocks. */ byte[] dataBlock; // Buffer for the data block byte dbAddress = 5; // Data block-address 5 string snr; // for serial number // ... Serial number e.g. take from Text field and store it in snr // ... data block e.g. take from Text field and store it in dataBlock // determine table index of the serial-number int idx = reader.FindTableIndex(0, ISO_TABLE, DATA_SNR, snr); if(idx < 0) return; // set serial-number for Addressed Mode </pre>

[Control byte] protocol	Example ⁸
	<pre> reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x24.Req.UID, snr); reader.SetData(OBID.ReaderCommand._0xB0.SUB_COMMAND, (byte)0x24); // Command Read Multiple Blocks reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x24.Req.MODE, (byte)0x00); // Mode-Byte reset reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x24.Req.Mode.ADR, (byte)0x01);// Addressed Mode reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x24.Req.NormAddrMode.DBN, (byte)0x01); // write one data block reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x24.Req.NormAddrMode.DB_ADR, dbAddress); // set data block address reader.SetTableData(idx, ISO_TABLE, DATA_BLOCK_SIZE, (byte)0x08); // set blocksize to e.g. 8 // write one data block (with blocksize of 8 bytes!) in the table reader.SetTableData(idx, ISO_TABLE, DATA_TxDB, ucDBAdr, dataBlock); reader.SendProtocol(0xB0); // Communication with reader/transponder </pre>

[Control byte] protocol	Example ⁸
<p>[0x24] Write Multiple Blocks (extended address mode)</p>	<pre> /* The example shows the [0x24] Write Multiple Block. In Addressed Mode an [0x01] Inventory must first be performed. Note: If [0x23] Read Multiple Blocks was not yet carried out, then the block size is preset to 4. But if the transponder in the read field supports another block size, this must first be set in the table for this transponder!! You can use GetTableData(..., DATA_IS_BLOCK_SIZE_SET) to check whether the block size was already read with [0x23] Read Multiple Blocks. */ byte[] dataBlock; // Buffer for the data block uint dbAddress = 5; // Data block-address 5 string snr; // for serial number string sPw; // for access password // ... Serial number e.g. take from Text field and store it in snr // ... data block e.g. take from Text field and store it in dataBlock // ... take password e. g. from text field // determine table index of the serial-number int idx = reader.FindTableIndex(0, ISO_TABLE, DATA_SNR, snr); if(idx < 0) return; // set serial-number for Addressed Mode reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x24.Req.UID, snr); reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x24.Req.ExtAddrMode.UID_LEN, snr.Length / 2); // length of UID in byte reader.SetData(OBID.ReaderCommand._0xB0.SUB_COMMAND, (byte)0x24); // Command Read Multiple Blocks reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x24.Req.MODE, (byte)0x00); // clear mode byte reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x24.Req.Mode.ADR, (byte)0x01); // addressed mode reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x24.Req.Mode.EXT_ADR, true); // extended addressed mode reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x24.Req.Mode.UID_LF, true); // length of UID != 8 reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x24.Req.ExtAddrMode.BANK, (byte)0x00); // clear bank nyte reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x24.Req.ExtAddrMode.Bank.NUMBER, (byte)0x03); // bank User Memory reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x24.Req.ExtAddrMode.Bank.ACCESS_ FLAG, true); // with access password reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x24.Req.ExtAddrMode.ACCESS_PW_L ENGTH, (byte)sPw.Length/2); // Len in bytes reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x24.Req.ExtAddrMode.ACCESS_PW, sPw); // password reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x24.Req.ExtAddrMode.DB_ADR, dbAddress); // datablock address reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x24.Req.ExtAddrMode.DBN, (byte)0x01); // write one data block reader.SetTableData(idx, ISO_TABLE, DATA_BLOCK_SIZE, (byte)0x08); // set blocksize to e.g. 8 // write one data block (with blocksize of 8 bytes!) in the table reader.SetTableData(idx, ISO_TABLE, DATA_TxDB, ucDBAdr, dataBlock); reader.SendProtocol(0xB0); // Communication with reader/transponder </pre>

[Control byte] protocol	Example ⁸
[0x25] Select	<pre> string snr; // for Serial-number // ... Serial number e.g. take from Text field and store it in snr // set Serial-number for Addressed Mode reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x25.Req.UID, snr); reader.SetData(OBID.ReaderCommand._0xB0.SUB_COMMAND, (byte)0x25); // Command Select reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x25.Req.MODE, (byte)0x00); // Mode-Byte reset reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x25.Req.Mode.ADR, (byte)0x01); // Addressed Mode reader.SendProtocol(0xB0); // Communication with reader/transponder </pre>
[0x25] Select mit Option Card Information für ISO14443 Transponder	<pre> string snr; // for Serial-number byte format = 0; // Format byte from response protocol // ... Serial number e.g. take from Text field and store it in snr // set Serial-number for Addressed Mode reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x25.Req.UID, snr); reader.SetData(OBID.ReaderCommand._0xB0.SUB_COMMAND, (byte)0x25); // Command Select reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x25.Req.MODE, (byte)0x00); // Mode-Byte reset reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x25.Req.Mode.ADR, (byte)0x01); // Addressed Mode reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x25.Req.Mode.CINF, true); // CINF-Flag reader.SendProtocol(0xB0); // Communication with reader/transponder // the Format byte is stored in TMPDATA_MEM reader.GetData(OBID.ReaderCommand._0xB0.SubCmd._0x25.Rsp.FORMAT; FORMAT); // Format // the Card Information is stored in TMPDATA_MEM beginning at Index 2048 // the structur and length of the Card Information according to the system manual // the principle access looks like this: // byte[] cardInfo; // int length = s. Systemhandbuch // reader.GetData(2048, cardInfo, length, TMPDATA_MEM); </pre>
[0x26] Reset to Ready	<pre> string snr; // for serial-number // ... Serial number e.g. take from Text field and store it in snr // set serial-number for Addressed Mode reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x26.Req.UID, snr); reader.SetData(OBID.ReaderCommand._0xB0.SUB_COMMAND, (byte)0x26); // Command Reset to Ready reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x26.Req.MODE, (byte)0x00); // Mode-Byte reset reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x26.Req.Mode.ADR, (byte)0x01); // Addressed Mode reader.SendProtocol(0xB0); // Communication with reader/transponder </pre>
[0x27] Write AFI	<pre> string snr; // for serial-number byte afi = 0; // for AFI // ... Serial number e.g. take from Text field and store it in snr // ... AFI e.g. take from Text field and store it in snr // set serial-number for Addressed Mode reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x27.Req.UID, snr); // determine table index of the serial-number </pre>

[Control byte] protocol	Example ⁸
	<pre> int idx = reader.FindTableIndex(0,ISO_TABLE, DATA_SNR, snr); if(idx < 0) return; // write AFI in table reader.SetTableData(idx, ISO_TABLE, DATA_AFI, afi); reader.SetData(OBID.ReaderCommand._0xB0.SUB_COMMAND, (byte)0x27);// Command Write AFI reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x27.Req.MODE, (byte)0x00);// reset Mode-Byte reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x27.Req.Mode.ADR, (byte)0x01);//Addr. Mode reader.SendProtocol(0xB0); // Communication with reader/transponder </pre>
[0x28] Lock AFI	<pre> string snr; // for serial-number // ... Serial number e.g. take from Text field and store it in snr // set serial-number for Addressed Mode reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x28.Req.UID, snr); reader.SetData(OBID.ReaderCommand._0xB0.SUB_COMMAND, (byte)0x28);// Command Lock AFI reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x28.Req.MODE, (byte)0x00);// Mode-Byte reset reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x28.Req.Mode.ADR, (byte)0x01);// Addr. Mode reader.SendProtocol(0xB0); // Communication with reader/transponder </pre>
[0x29] Write DSFID	<pre> string snr; // for serial number byte dsfid = 0; // for DSFID // ... Serial number e.g. take from Text field and store it in snr // ... dsfid e.g. take from Text field and store it in dsfid // set serial-number for Addressed Mode reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x29.Req.UID, snr); // determine table index of the serial number int idx = reader.FindTableIndex(0, ISO_TABLE, DATA_SNR, snr); if(idx < 0) return; // write DSFID in table reader.SetTableData(idx, ISO_TABLE, DATA_DSfid, dsfid); reader.SetData(OBID.ReaderCommand._0xB0.SUB_COMMAND, (byte)0x29);// Command Write DSFID reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x29.Req.MODE, (byte)0x00); // Mode-Byte reset reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x29.Req.Mode.ADR, (byte)0x01); // Addressed Mode reader.SendProtocol(0xB0); // Communication with reader/transponder </pre>
[0x2A] Lock DSFID	<pre> string snr; // for Serial-number // ... Serial number e.g. take from Text field and store it in snr // set Serial-number for Addressed Mode reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x2A.Req.UID, snr); reader.SetData(OBID.ReaderCommand._0xB0.SUB_COMMAND, (byte)0x2A);// Command Lock DSFID reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x2A.Req.MODE, (byte)0x00); // Mode-Byte reset reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x2A.Req.Mode.ADR, (byte)0x01); // Addr. Mode reader.SendProtocol(0xB0); // Communication with reader/transponder </pre>

[Control byte] protocol	Example ⁸
[0x2B] Get System Information	<pre> byte dsfid = 0; // for DSFID byte afi = 0; // for AFI byte[] ucMemSize = {0, 0}; // for memory size byte icRef = 0; // for IC-Reference string snr; // for serial number // ... Serial number e.g. take from Text field and store it in snr // set Serial-number for Addressed Mode reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x2B.Req.UID, snr); reader.SetData(OBID.ReaderCommand._0xB0.SUB_COMMAND, (byte)0x2B); // Command Get System Information reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x2B.Req.MODE, (byte)0x00); // Mode- Byte reset reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x2B.Req.Mode.ADR, (byte)0x01); // Addressed Mode reader.SendProtocol(0xB0); // Communication with reader/transponder // all transponder data are in the table // first determine the table index of the serial number int idx = reader.FindTableIndex(0, ISO_TABLE, DATA_SNR, snr); if(idx < 0) return; // take AFI reader.GetTableData(idx, ISO_TABLE, DATA_AFI, out afi); // ... do something with AFI // ... get all other data with the same procedure </pre>
[0x2C] Get Multiple Block Security Status	<pre> byte secStatus; // for Security Status string snr; // für Serial number // ... Serial number e.g. take from Text field and store it in snr // set Serial-number for Addressed Mode reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x2C.Req.UID, snr); reader.SetData(FEDM_ISC_TMP_B0_REQ_DBN, (byte)0x05); // 5 Data blocks reader.SetData(FEDM_ISC_TMP_B0_REQ_DB_ADR, (byte)0x00); // set 1st data block address reader.SetData(OBID.ReaderCommand._0xB0.SUB_COMMAND, (byte)0x2C); // Command Get Multiple Block Security Status reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x2C.Req.MODE, (byte)0x00); // Mode- Byte reset reader.SetData(OBID.ReaderCommand._0xB0.SubCmd._0x2C.Req.Mode.ADR, (byte)0x01); // Addr. Mode reader.SendProtocol(0xB0); // Communication with reader/transponder // all transponder data are in the table // first determine the table index of the serial number int idx = reader.FindTableIndex(0, ISO_TABLE, DATA_SNR, snr); if(idx < 0) return; // get the security status from block 0..4 for(int cnt=0; cnt<5; ++cnt) { reader.GetTableData(idx, ISO_TABLE, DATA_SEC_STATUS, cnt, out secStatus); // ... do something with secStatus } </pre>
[0xA0] Read Config Block only for I-Code 1	<pre> byte[] configBlock; // Buffer for a data block (blocksize is always 4) byte cbAddress = 0; // Data block-Address 0 string snr; // for serial number // ... Serial number e.g. take from Text field and store it in snr // set serial number for Addressed Mode </pre>

[Control byte] protocol	Example ⁸
	<pre>reader.SetData(FEDM_ISC_TMP_B0_REQ_UID, snr); reader.SetData(OBID.ReaderCommand._0xB0.SUB_COMMAND, (byte)0xA0); // Command Read Configuration Block reader.SetData(FEDM_ISC_TMP_B0_MODE, (byte)0x00); // Mode-Byte reset reader.SetData(FEDM_ISC_TMP_B0_MODE_ADR, (byte)0x01); // Addressed Mode reader.SetData(FEDM_ISC_TMP_B0_REQ_CB_ADR, cbAddress); // set data block address reader.SendProtocol(0xB0); // Communication with reader/transponder // all transponder data are in the table // first determine the table index of the serial number int idx = reader.FindTableIndex(0, ISO_TABLE, DATA_SNR, snr); if(idx < 0) return; // take the data block reader.GetTableData(idx, ISO_TABLE, DATA_RxCB, cdAddress, out configBlock); // ... do something with the data block</pre>

[Control byte] protocol	Example ⁸
[0xA1] Write Config Block only for I-Code 1	<pre> /* <u>Attention</u>: With this ISO Command you can change the configuration of the transponders and this can change the function of the transponder and so the transponder can be useless!! */ byte[] configBlock; // buffer for a data block (blocksize is always 4) byte cbAddress = 0; // Data block address 0 string snr; // for serial number // ... Serial number e.g. take from text field and store it in snr // ...data block e.g. take from text field and store it in configBlock // first determine the table index of the serial number int idx = reader.FindTableIndex(0, ISO_TABLE, DATA_SNR, snr); if(idx < 0) return; // set serial number for Addressed Mode reader.SetData(FEDM_ISC_TMP_B0_REQ_UID, snr); reader.SetData(OBID.ReaderCommand._0xB0.SUB_COMMAND, (byte)0xA1); // Command Write Multiple Block reader.SetData(FEDM_ISC_TMP_B0_MODE, (byte)0x00); // Mode byte reset reader.SetData(FEDM_ISC_TMP_B0_MODE_ADR, (byte)0x01); // Addr. Mode reader.SetData(FEDM_ISC_TMP_B0_REQ_CB_ADR, cbAddress); // set data block address // write a data block intoTable reader.SetTableData(idx, ISO_TABLE, DATA_TxCB, cbAddress, configBlock); reader.SendProtocol(0xB0); // Communication with reader/transponder </pre>

8.2.3.Examples for using the ISO table with [0xB3] Commands

[Control byte] protocol	Example ⁹
<p>[0x18] Kill</p> <p>for UHF-Transponder:</p> <ul style="list-style-type: none"> - EPC Class1 Gen1 - EPC Class1 Gen2 	<pre> /* <u>Attention</u>: with this command transponders are destroyed irretrievably! string sEpc; // for EPC string sPw; // for Kill Password byte cEpcLen = 0; // length of EPC in byte byte cPwLen = 0; // length of Kill Password // ... EPC e.g. take from text field and store it in epc, dito with the length // ... Kill Password e.g. take from text field and store it in pw, dito with the length // determine table index of the EPC int idx = reader.FindTableIndex(0, ISO_TABLE, DATA_SNR, sEpc); // set EPC for addressed mode reader.SetData(FEDM_ISC_TMP_B3_REQ_EPC, sEpc); reader.SetData(FEDM_ISC_TMP_B3_REQ_EPC_LEN, cEpcLen); // length of EPC reader.SetData(FEDM_ISC_TMP_B3_CMD, (byte)0x18); // Command Kill reader.SetData(FEDM_ISC_TMP_B3_MODE, (byte)0x00); // reset mode byte reader.SetData(FEDM_ISC_TMP_B3_MODE_ADR, (byte)0x01); // addressed mode reader.SetData(FEDM_ISC_TMP_B3_MODE_EPC_LF, true); // EPC length flag reader.SetData(FEDM_ISC_TMP_B3_KILL_PW_LENGTH, cPwLen); // length of Kill Password reader.SetData(FEDM_ISC_TMP_B3_KILL_PW, sPw); // Kill Password reader.SendProtocol(0xB3); // communication with Reader/Transponder </pre>
<p>[0x22] Lock Multiple Blocks</p> <p>for UHF-Transponder:</p> <ul style="list-style-type: none"> - EPC Class1 Gen1 - EPC Class1 Gen2 	<pre> // <u>Attention</u>: with this ISO Command all data blocks will be locked irretrievably! string sEpc; // for EPC string sLockData; // for Lock Data string sPw; // for Access Password byte cEpcLen = 0; // length of EPC in byte byte cTrType = 0; // transponder type byte cLockDataLen = 0; // length of Lock Data in byte byte cPwLen = 0; // length of Access Password in byte // ... EPC e.g. take from text field and store it in epc, dito with the length // ... Lock Data e.g. take from text field and store it in lockData, dito with the length // ... Kill Password e.g. take from text field and store it in pw, dito with the length // determine table index of the EPC int idx = reader.FindTableIndex(0, ISO_TABLE, DATA_SNR, sEpc); // determine the transponder type reader.GetTableData(idx, ISO_TABLE, DATA_TRTYPE, out cTrType); // set EPC for addressed mode reader.SetData(FEDM_ISC_TMP_B3_REQ_EPC, sEpc); reader.SetData(FEDM_ISC_TMP_B3_REQ_EPC_LEN, cEpcLen); // length of EPC reader.SetData(FEDM_ISC_TMP_B3_CMD, (byte)0x22); // Command Lock reader.SetData(FEDM_ISC_TMP_B3_MODE, (byte)0x00); // reset mode byte reader.SetData(FEDM_ISC_TMP_B3_MODE_ADR, (byte)0x01); // addressed mode reader.SetData(FEDM_ISC_TMP_B3_MODE_EPC_LF, true); // EPC length flag reader.SetData(FEDM_ISC_TMP_B3_REQ_TR_TYPE, cTrType); // transponder type reader.SetData(FEDM_ISC_TMP_B3_LOCK_DATA_LENGTH, cLockDataLen); // length of // Lock Data </pre>

⁹ all examples in C#

[Control byte] protocol	Example ⁹
	<pre> reader.SetData(FEDM_ISC_TMP_B3_LOCK_DATA, sLockData); // Lock Data reader.SetData(FEDM_ISC_TMP_B3_ACCESS_PW_LENGTH, cPwLen); // length of Access // Passwort if(cPwLen > 0) reader.SetData(FEDM_ISC_TMP_B3_ACCESS_PW, sPw); // Access Password reader.SendProtocol(0xB3); // communication with Reader/Transponder </pre>
<p>[0x24] Write Multiple Blocks</p> <p>for UHF-Transponder: - EPC Class1 Gen2</p>	<pre> /* The example shows the [0x24] Write Multiple Block. In Addressed Mode an [0x01] Inventory must first be performed. <u>Note:</u> If [0x23] Read Multiple Blocks was not yet carried out, then the block size is preset to 4. But if the transponder in the read field supports another block size, this must first be set in the table for this transponder!! You can use GetTableData(..., DATA_IS_BLOCK_SIZE_SET) to check whether the block size was already read with [0x23] Read Multiple Blocks.*/ byte[][] cDB; // buffer for Data (1. dimension for block number, 2. dimension für data) string sEpc; // for EPC string sPw; // for optional Access Password byte cEpcLen = 0; // length of EPC in byte byte cPwLen = 0; // length of optional Access Password // ... EPC e.g. take from Text field and store it in sEpc // ... Access Password e.g. take from text field and store it in sPw, dito with the length // ... data block e.g. take from Text field and store it in cDB // determine table index of the EPC int idx = reader.FindTableIndex(0, ISO_TABLE, DATA_SNR, sEpc); // set EPC for addressed mode reader.SetData(FEDM_ISC_TMP_B3_REQ_UID, sEpc); reader.SetData(FEDM_ISC_TMP_B3_REQ_EPC_LEN, cEpcLen); // length of EPC reader.SetData(FEDM_ISC_TMP_B3_CMD, (byte)0x24); // Command Read Multiple // Blocks reader.SetData(FEDM_ISC_TMP_B3_MODE, (byte)0x00); // reset mode byte reader.SetData(FEDM_ISC_TMP_B3_MODE_ADR, (byte)0x01); // addressed mode reader.SetData(FEDM_ISC_TMP_B3_MODE_EPC_LF, true); // EPC length flag reader.SetData(FEDM_ISC_TMP_B3_MODE_EXT_ADR, true); // extended address mode reader.SetData(FEDM_ISC_TMP_B3_BANK_ACCESS_FLAG, true); // Access Password flag reader.SetData(FEDM_ISC_TMP_B3_BANK_BANK_NR, (byte)0x01); // EPC bank number reader.SetData(FEDM_ISC_TMP_B3_REQ_DBN, (byte)0x06); // six data blocks to write reader.SetData(FEDM_ISC_TMP_B3_REQ_DB_ADR_EXT, (uint)0); // first data block address reader.SetData(FEDM_ISC_TMP_B3_REQ_DB_SIZE, (byte)0x02); // block size for command // set block size in table reader.SetTableData(idx, ISO_TABLE, DATA_BLOCK_SIZE, (byte)2); // write data blocks in table for(int iAdr=0; iAdr<6; ++iAdr) reader.SetTableData(idx, ISO_TABLE, DATA_TxDB, iAdr, cDB[iAdr]); reader.SendProtocol(0xB3); // communication with Reader/Transponder </pre>

8.2.4. Commands for Buffered Read Mode

[Control byte] protocol	Example ¹⁰
[0x21] Read Buffer	<pre> // this sample shows the reading of data sets with serial number, data block and Timer-value byte dataSets = 1; // number requested data sets byte recSets = 0; // number data sets in receive protocol byte[] dataBlock; // buffer for a data block FelscReaderTime time = 0; // for Timer-value string snr; // for serial number bool snrFlag = false; // flag for serial number in dataset bool dbFlag = false; // flag for data block in dataset bool timerFlag = false; // flag for Timer in dataset FedmBrmTableItem item; // a table entry with data for one transponder reader.SetData(FedmIscReaderID.FEDM_ISCLR_TMP_BRM_SETS, dataSets); reader.SendProtocol((byte)0x21); // read data blocks from transponder with Buffered Read Mode reader.GetData(FedmIscReaderID.FEDM_ISCLR_TMP_BRM_TRDATA_SNR, out snrFlag); reader.GetData(FedmIscReaderID.FEDM_ISCLR_TMP_BRM_TRDATA_DB, out dbFlag); reader.GetData(FedmIscReaderID.FEDM_ISCLR_TMP_BRM_TRDATA_TIME, out timerFlag); reader.GetData(FedmIscReaderID.FEDM_ISCLR_TMP_BRM_RECSETS, out recSets); // All transponder data content in the table for(int cnt=0; cnt< reader.GetTableLength(FedmIscReaderConst.BRM_TABLE); cnt++) { item = (FedmBrmTableItem) reader.GetTableItem(cnt, FedmIscReaderConst.BRM_TABLE); if(snrFlag) // get serial number item.GetData(FedmIscReaderConst.DATA_SNR, out snr); if(dbFlag) // get data block 1 item.GetData(FedmIscReaderConst.DATA_RxDB, dataBlock); if(timerFlag) // get Timer-value time = item.GetReaderTime(); } </pre>
[0x22] Read Buffer	<pre> // this sample shows the reading of Data sets with serial number, data block, Timer-value, Date and // antenna number uint uiDataSets = 1; // Number requested Data sets uint uiRecSets = 0; // number data sets in receive protocol byte cAnt; // antenna number byte cInput = 0; // Input-Byte byte cState = 0; // Status-Byte FelscReaderTime time = 0; // for Timer-value byte cSize; // for blocksize uint uiDBN = 0; // for number of data blocks string sSnr; // for serial number string sDB; // for data blocks bool bSNR = false; // flag for serial number in data record bool bDB = false; // flag for data block in data record bool bANT = false; // flag for antenna number in data record bool bTime = false; // flag for time in data record </pre>

¹⁰ all examples in C#

[Control byte] protocol	Example ¹⁰
	<pre> bool bDate = false; // flag for date in data record bool bExt = false; // EXTENSION flag (in TR-DATA1): signals, that a second TR-DATA // byte is following, where additional flags continues the definition of a // data set bool bInput = FALSE; // flag (in TR-DATA2) for input and status byte in data set FedmBrmTableItem item; // a table entry with data for one transponder reader.SetData(OBID.ReaderCommand._0x22.Req.DATA_SETS, uiDataSets); reader.SendProtocol(0x22); // read data blocks from transponder with Buffered Read Mode reader.GetData(OBID.ReaderCommand._0x22.Req.TrData1.UID, out bSNR); reader.GetData(OBID.ReaderCommand._0x22.Req.TrData1.DB, out bDB); reader.GetData(OBID.ReaderCommand._0x22.Req.TrData1.ANT, out bANT); reader.GetData(OBID.ReaderCommand._0x22.Req.TrData1.TIME, out bTime); reader.GetData(OBID.ReaderCommand._0x22.Req.TrData1.DATE, out bDate); reader.GetData(OBID.ReaderCommand._0x22.Req.TrData1.EXT, out bExt); reader.GetData(OBID.ReaderCommand._0x22.Req.TrData2.INPUT, out bInput); reader.GetData(OBID.ReaderCommand._0x22.Rsp.DATA_SETS, out uiRecSets); // All transponder data content in the table for(int iCnt=0; iCnt< reader.GetTableLength(FEDM_ISC_BRM_TABLE); iCnt++) { item = (FedmBrmTableItem) reader.GetTableItem(cnt, BRM_TABLE); if(bSNR) // get serial number item.GetData(cnt, DATA_SNR, out sSnr); if(bDB) // get all data blocks { // get number of data blocks item.GetData(iCnt, DATA_DBN, out uiDBN); // get the blocksize item.GetData(iCnt, DATA_BLOCK_SIZE, out cSize); // get data blocks for(int i=0; i<uiDBN; ++i) { item.GetData(iCnt, DATA_RxDB, i, out sDB); // do anything with the data blocks } } if(bANT) // get antenna number item.GetData(iCnt, DATA_ANT_NR, out cAnt); if(bTime bDate) // get date and/or time time = item.GetReaderTime(); if(bExt && bInput) // get input and status byte { GetData(DATA_INPUT, out cInput); GetData(DATA_STATE, out cState); } } </pre>

8.3. Commands for function unit

SendProtocol is of key importance to the protocol transfer. For this reason an example is shown for each control byte which is intended to show which data are to be stored in the data container with which access constants before each protocol transfer and which data are available after the protocol transfer.

All the access constants are listed in the structure FedmlscFunctionUnitID and should be studied carefully in conjunction with the explanation of the protocol data found in the system manual.

For reasons of clarity the processing of the return values of the functions is not shown here. Of course it should always be included in applications.

In the examples below it is assumed that the Function Unit class **FedmlscFunctionUnit** and the interfaces **FedmlscFunctionUnitID** are incorporated:

[Control byte] Protocol		Example ¹¹
ID ISC.DAT	[0xC0] Get Firmware Version	<pre>string sFirmware; // buffer for firmware informations fu.SendProtocol(0xC0); fu.GetData(FEDM_ISC_FU_TMP_SOFTVER, out sFirmware);</pre>
	[0xC1] CPU Reset	<pre>fu.SendProtocol(0xC1);</pre>
	[0xC2] Set Capacities	<pre>fu.SetData(FEDM_ISC_FU_TMP_DAT_ANT_VAL_C1, (byte)0xAB); // capacity 1 fu.SetData(FEDM_ISC_FU_TMP_DAT_ANT_VAL_C2, (byte)0x9F); // capacity 2 fu.SendProtocol(0xC2);</pre>
	[0xC3] Get Antenna Values	<pre>string sAntValues; // buffer for tuning values fu.SendProtocol(0xC3); fu.GetData(FEDM_ISC_FU_TMP_DAT_ANT_VAL, out sAntValues);</pre>
	[0xC4] Set Outputs	<pre>fu.SetData(FEDM_ISC_FU_TMP_DAT_OUT, (byte)1); // switch output 1 fu.SendProtocol(0xC4);</pre>
	[0xC5] Re-Tuning	<pre>fu.SendProtocol(0xC5);</pre>
	[0xC6] Start Tuning	<pre>fu.SendProtocol(0xC6);</pre>
	[0xC8] Store Settings	<pre>fu.SendProtocol(0xC8);</pre>
	[0xC9] Detect	<pre>fu.SendProtocol(0xC9);</pre>
	[0xCA] Set Address	<pre>byte cAdr = 2; // new address fu.SetData(FEDM_ISC_FU_TMP_DAT_NEW_ADR, cAdr); // new address for function unit fu.SendProtocol(0xCA); // new address becomes valid fu.SetData(FEDM_ISC_FU_TMP_DAT_ADR, cAdr); // set new address for communication</pre>
	[0xCB] Set Mode	<pre>fu.SetData(FEDM_ISC_FU_TMP_DAT_MODE, (byte)1); // mode 1 fu.SendProtocol(0xCB);</pre>
ISC.Z	[0xDC] Detect	<pre>fu.SendProtocol(0xDC);</pre>

¹¹ all examples in C#

[Control byte] Protocol		Example ¹¹
ID ISC.ANT.UMUX	[0xDD] Select Channel	<pre>fu.SetData(FEDM_ISC_FU_TMP_MUX_OUT_CH1, (byte)1); // set output 1 for input 1 fu.SetData(FEDM_ISC_FU_TMP_MUX_OUT_CH2, (byte)8); // set output 8 for input 2 fu.SendProtocol(0xDD);</pre>
	[0xDE] CPU Reset	<pre>fu.SendProtocol(0xDE);</pre>
	[0xDF] Get Firmware Version	<pre>string sFirmware; // buffer for firmware informations fu.SendProtocol(0xDF); fu.GetData(FEDM_ISC_FU_TMP_SOFTVER, out sFirmware);</pre>
	[0xDC] Detect/Get Power	<pre>byte[] Power = new byte[5]; // buffer for Power Information byte UMuxState = 0; // statusbyte of response fu.SetData(FEDM_ISC_FU_TMP_FLAGS, (byte)0); // set always to 0 fu.SendProtocol(0xDC); fu.GetData(FEDM_ISC_FU_TMP_UMUX_POWER, Power); fu.GetData(FEDM_ISC_FU_TMP_UMUX_LAST_STATE, out UMuxStatus);</pre>
	[0xDD] Select Channel	<pre>byte UMuxState = 0; // statusbyte of response fu.SetData(FEDM_ISC_FU_TMP_FLAGS, (byte)0); // set always to 0 fu.SetData(FEDM_ISC_FU_TMP_MUX_OUT_CH1, (byte)1); // select output 1 fu.SendProtocol(0xDD); fu.GetData(FEDM_ISC_FU_TMP_UMUX_LAST_STATE, out UMuxStatus);</pre>
	[0xDE] CPU Reset	<pre>byte UMuxState = 0; // statusbyte of response fu.SetData(FEDM_ISC_FU_TMP_FLAGS, (byte)0); // set always to 0 fu.SendProtocol(0xDE); fu.GetData(FEDM_ISC_FU_TMP_UMUX_LAST_STATE, out UMuxStatus);</pre>
	[0xDF] Get Firmware Version	<pre>byte[] Firmware = new byte[7]; // buffer for Firmware Information byte UMuxState = 0; // statusbyte of response fu.SetData(FEDM_ISC_FU_TMP_FLAGS, (byte)0); // set always to 0 fu.SendProtocol(0xDF); fu.GetData(FEDM_ISC_FU_TMP_SOFTVER, out Firmware); fu.GetData(FEDM_ISC_FU_TMP_UMUX_LAST_STATE, out UMuxStatus);</pre>

9.Example for using TagHandler classes

```
using OBID;
using OBID.TagHandler;

...
FedmIscReaderReader = new FedmIscReader();
Reader.SetTableSize(10); // max 10 Transponder for each Inventory

...
// connection to the Reader
Reader.ConnectUSB(0);

...
int back = 0;
int tagDriver = 0; // set to 9 for MIFARE DESFire
byte BlockSize = 0;
byte[] Data = null;
Dictionary<string, FedmIscTagHandler> TagList;
Dictionary<string, FedmIscTagHandler>.ValueCollection listTagHandler;
FedmIscTagHandler TagHandler = null;

// Inventory command with standard options
TagList = Reader.TagInventory(true, 0x00, 1);

if (TagList.Count > 0)
{
    listTagHandler = TagList.Values;
    foreach (FedmIscTagHandler tagHandler in listTagHandler)
    {
        if (tagHandler != null)
        {
            TagHandler = tagHandler;

            // select Transponder:
            // necessary for ISO 14443 (optional: set tagDriver (s. System Manual of Reader))
            // optional for ISO 15693
            // not for EPC Class 1 Gen 2
            TagHandler = Reader.TagSelect(TagHandler, tagDriver);

            else if (TagHandler is FedmIscTagHandler_ISO15693)
            {
                FedmIscTagHandler_ISO15693 thIso = (FedmIscTagHandler_ISO15693)TagHandler;

                // read datablocks and write same data back
                back = thIso.ReadMultipleBlocksWithSecStatus(4, 4, out BlockSize, out Data);
                back = thIso.WriteMultipleBlocks(4, 4, 4, Data);
            }
            else if (TagHandler is FedmIscTagHandler_ISO14443_4_MIFARE_DESFire)
            {
                FedmIscTagHandler_ISO14443_4_MIFARE_DESFire thIso =
                    (FedmIscTagHandler_ISO14443_4_MIFARE_DESFire)TagHandler;

                // read version information
                // use of the internal Interface IFlexSoftCrypto
                back = thIso.IFlexSoftCrypto.GetVersion((byte)0, out Data);
            }
            else if (TagHandler is FedmIscTagHandler_EPC_Class1_Gen2)
            {

```

```
FedmIsctagHandler_EPC_Class1_Gen2 thGen2 = (FedmIsctagHandler_EPC_Class1_Gen2)TagHandler;  
  
// write new EPC to Transponder (without Password)  
thGen2.WriteEPC("0102030405060708090A0B0C", "");  
}  
}
```

10. Appendix

10.1.List of error codes

All listed error codes are located in the structure Fedm.

Error constant	Value	Description
MODIFIED	1	Indicates a modification of a container. This is not an error.
OK	0	No error
ERROR_BLOCK_SIZE	-101	Block size in the access constant is incorrect
ERROR_BIT_BOUNDARY	-102	Bit boundary in the access constant is incorrect
ERROR_BYTE_BOUNDARY	-103	Byte boundary in the access constant is incorrect
ERROR_ARRAY_BOUNDARY	-104	Array boundary of a data container was exceeded
ERROR_BUFFER_LENGTH	-105	Length of the data buffer is insufficient
ERROR_PARAMETER	-106	Unknown transfer parameter
ERROR_STRING_LENGTH	-107	Transferred string is too long
ERROR_ODD_STRING_LENGTH	-108	Transferred string contains an odd number of characters
ERROR_NO_DATA	-109	No data in the protocol
ERROR_NO_READER_HANDLE	-110	No reader handle set
ERROR_NO_PORT_HANDLE	-111	No port handle set
ERROR_UNKNOWN_CONTROL_BYTE	-112	Unknown control byte
ERROR_UNKNOWN_MEM_ID	-113	Unknown memory ID
ERROR_UNKNOWN_POLL_MODE	-114	Unknown poll mode
ERROR_NO_TABLE_DATA	-115	No data in a table
ERROR_UNKNOWN_ERROR_CODE	-116	Unknown error code
ERROR_UNKNOWN_COMMAND	-117	Unknown command
ERROR_UNSUPPORTED	-118	No support for this parameter or function
ERROR_NO_MORE_MEM	-119	No more program memory available
ERROR_NO_READER_FOUND	-120	No reader found
ERROR_NULL_POINTER	-121	The transferred pointer is NULL
ERROR_UNKNOWN_READER_TYPE	-122	Unknown reader type
ERROR_UNSUPPORTED_READER_TYPE	-123	The Function doesn't support this reader type

Error constant	Value	Description
ERROR_UNKNOWN_TABLE_ID	-124	Unknown table constant
ERROR_UNKNOWN_LANGUAGE	-125	Unknown language constant
ERROR_NO_TABLE_SIZE	-126	The table has the size 0
ERROR_SENDBUFFER_OVERFLOW	-127	The Sendbuffer is full
ERROR_VERIFY	-128	Data are not equal
ERROR_OPEN_FILE	-129	File open error
ERROR_SAVE_FILE	-130	File save error
ERROR_UNKNOWN_TRANSPONDER_TYPE	-131	Unknown transponder type
ERROR_READ_FILE	-132	Read file error
ERROR_WRITE_FILE	-133	Write file error
ERROR_UNKNOWN_EPC_TYPE	-134	Unknown EPC-Typ
ERROR_UNSUPPORTED_PORT_DRIVER	-135	Function does not support the active communication driver
ERROR_UNKNOWN_ADDRESS_MODE	-136	Unknown address mode
ERROR_ALREADY_CONNECTED	-137	Reader object is already connected with a communication port
ERROR_NOT_CONNECTED	-138	Reader object is not connected with a communication port
ERROR_NO_MODULE_HANDLE	-139	No module handle found
ERROR_EMPTY_MODULE_LIST	-140	The module list is empty
ERROR_MODULE_NOT_FOUND	-141	Module not found in module list
ERROR_DIFFERENT_OBJECTS	-142	Runtime objects are different
ERROR_NOT_AN_EPC	-143	IDD of transponder is not an EPC
ERROR_OLD_LIB_VERSION	-144	Old library file (error code for Java/.NET-Libraries)
ERROR_WRONG_READER_TYPE	-145	Wrong reader type
ERROR_CRC	-146	CRC error in file
ERROR_CFG_BLOCK_PREVIOUSLY_NOT_READ	-147	Configuration block must be read first
ERROR_UNSUPPORTED_CONTROLLER_TYPE	-148	Unsupported controller type
ERROR_VERSION_CONFLICT	-149	Version conflict with one or more dependent libraries
ERROR_UNSUPPORTED_NAMESPACE	-150	The namespace is not supported by the reader type
ERROR_TASK_STILL_RUNNING	-151	Asynchronous task is still running
ERROR_TAG_HANDLER_NOT_IDENTIFIED	-152	TagHandler type could not be identified
ERROR_UNVALID_IDD_LENGTH	-153	Value of IDD-Length is out of range
ERROR_UNVALID_IDD_FORMAT	-154	Value of IDD-Format is out of range
ERROR_UNKNOWN_TAG_HANDLER_TYPE	-155	Unknown TagHandler type

Error constant	Value	Description
ERROR_UNSUPPORTED_TRANSPONDER_TYPE	-156	Transponder- or Chip-Type is not supportet
ERROR_CONNECTED_WITH_OTHER_MODULE	-157	Only TCP/IP: a connection to the same Reader still established by another Reader module.
ERROR_INVENTORY_NO_TID_IN_UID	-158	Inventory with return of UID = EPC + TID, but TID is missing
XML_ERROR_NO_XML_FILE	-200	File is not a XML document
XML_ERROR_NO_OBID_TAG	-201	File contains no element 'OBID'
XML_ERROR_NO_CHILD_TAG	-202	No sub-element found
XML_ERROR_TAG_NOT_FOUND	-203	Element not in the document
XML_ERROR_DOC_NOT_WELL_FORMED	-204	XML document not well-formed
XML_ERROR_NO_TAG_VALUE	-205	No content of element found
XML_ERROR_NO_TAG_ATTRIBUTE	-206	No attribute found
XML_ERROR_DOC_FILE_VERSION	-207	Unvalid document version
XML_ERROR_DOC_FILE_FAMILY	-208	The Document is for another reader family
XML_ERROR_DOC_FILE_TYPE	-209	Wrong file type
XML_ERROR_WRONG_CONTROLLER_TYPE	-210	Wrong controller type
XML_ERROR_WRONG_MEM_BANK_TYPE	-211	Wrong memory bank

10.2.Supported OBID® Readers

Reader	Notes
ID ISC.M02	
ID ISC.MR/PR100	all communication ports
ID ISC.PRH100/PRH101 / PRH102	all communication ports
ID ISC.MR/PR101	all communication ports
ID ISC.MR102	all communication ports
ID ISC.PRH102	all communication ports
ID ISC.PRHD102	all communication ports
ID ISC.MR200	all communication ports
ID ISC.LR200	
ID ISC.LR1002	all communication ports
ID ISC.LR2000	all communication ports
ID ISC.LR2500-A	all communication ports
ID ISC.LR2500-B	all communication ports
ID ISC.MU02	
ID ISC.MRU102	all communication ports
ID ISC.MRU200	all communication ports
ID ISC.LRU1000	all communication ports
ID ISC.LRU2000	all communication ports
ID ISC.LRU3000	all communication ports
ID CPR.02	
ID CPR.M02	all communication ports
ID CPR.04	all communication ports
ID CPR30.xx	all communication ports
ID CPR40.xx	all communication ports
ID CPR44.xx	all communication ports
ID CPR46.xx	all communication ports
ID CPR50.xx	all communication ports
ID CPR52.xx	all communication ports
ID MAX50.xx	all communication ports

10.3. Supported Transponders

The support of transponders depends on the implemented reader firmware. Please refer to the system manual of the reader.

The list below collects the transponder types, which are well-established during the development time of the library.

Transponder	Value	Notes
I-CODE 1	0x00	HF-Transponder
Tag-it	0x01	HF-Transponder
ISO15693	0x03	HF-Transponder
ISO14443-A	0x04	HF-Transponder
ISO14443-B	0x05	HF-Transponder
EPC	0x06	HF-Transponder (EPC-Types 1..4)
I-CODE UID	0x07	HF-Transponder
Jewel	0x08	HF-Transponder
ISO 18000-3M3	0x09	HF-Transponder
STMicroelectronics SR176	0x0A	HF-Transponder
STMicroelectronics SRIxx	0x0B	HF-Transponder
Microchip MCRFxxx	0x0C	HF-Transponder
Innovatron (ISO 14443B')	0x10	HF-Transponder
ASK CTx	0x11	HF-Transponder
ISO18000-6-A	0x80	UHF-Transponder
ISO18000-6-B	0x81	UHF-Transponder
EM4222	0x83	UHF-Transponder
EPC Class1 Generation 2	0x84	UHF-Transponder
EPC Class0/0+	0x88	UHF-Transponder
EPC Class1 Generation 1	0x89	UHF-Transponder

10.4. TCP Status

Information concerning the status can be found with the Internet when searching for *Transmission Control Protocol*

TCP-Status	Value
CLOSED	1
LISTEN	2
SYN_SENT	3
SYN_RCVD	4
ESTABLISHED	5
FIN_WAIT1	6
FIN_WAIT2	7
CLOSE_WAIT	8
CLOSING	9
LAST_ACK	10
TIME_WAIT	11

10.5. List of constants

All constants listed here are defined in **FedmlSCReaderConst**.

10.5.1. General constants

Constant	Description
TYPE_...	Reader Type according to the protocol [0x65] Software Version
NAME_...	Reader Name according to the readers system manual
TR_TYPE_...	Transponder Type according to the appendix of the readers system manual
EPC_TYPE_...	EPC-Type; (EPC = Electronic Product Code)
FU_TYPE_...	Type of function units (see class FedmlscFunctionUnit)

10.5.2. Constants for tableID

Constant	Description
BRM_TABLE	Table-ID for BRM-Table
ISO_TABLE	Table-ID für ISO-Table

10.5.3. Constants for dataID

Constant	Description/Use																																
DATA_TRTYPE	Transponder type																																
<table><tr><th>BRM-Table</th><th>ISO-Table</th></tr><tr><td>X</td><td>X</td></tr></table>	BRM-Table	ISO-Table	X	X	<table><tr><th></th><th>bool</th><th>byte</th><th>byte[]</th><th>int</th><th>long</th><th>string</th></tr><tr><td>GetTableData</td><td></td><td>X</td><td>X</td><td>X</td><td></td><td>X</td></tr><tr><td>SetTableData</td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>FindTableIndex</td><td></td><td>X</td><td></td><td>X</td><td></td><td></td></tr></table>		bool	byte	byte[]	int	long	string	GetTableData		X	X	X		X	SetTableData							FindTableIndex		X		X		
BRM-Table	ISO-Table																																
X	X																																
	bool	byte	byte[]	int	long	string																											
GetTableData		X	X	X		X																											
SetTableData																																	
FindTableIndex		X		X																													
DATA_SNR	Serial number																																
<table><tr><th>BRM-Table</th><th>ISO-Table</th></tr><tr><td>X</td><td>X</td></tr></table>	BRM-Table	ISO-Table	X	X	<table><tr><th></th><th>bool</th><th>byte</th><th>byte[]</th><th>int</th><th>long</th><th>string</th></tr><tr><td>GetTableData</td><td></td><td></td><td>X</td><td></td><td>X</td><td>X</td></tr><tr><td>SetTableData</td><td></td><td></td><td></td><td></td><td>X</td><td>X</td></tr><tr><td>FindTableIndex</td><td></td><td></td><td></td><td></td><td>X</td><td>X</td></tr></table>		bool	byte	byte[]	int	long	string	GetTableData			X		X	X	SetTableData					X	X	FindTableIndex					X	X
BRM-Table	ISO-Table																																
X	X																																
	bool	byte	byte[]	int	long	string																											
GetTableData			X		X	X																											
SetTableData					X	X																											
FindTableIndex					X	X																											

Constant	Description/Use																																
DATA_RxDB DATA_RxDB_EPC_BANK DATA_RxDB_TID_BANK DATA_RxDB_RES_BANK <table><tr><th>BRM-Table</th><th>ISO-Table</th></tr><tr><td>X</td><td>X</td></tr></table>	BRM-Table	ISO-Table	X	X	Data blocks from receive protocol Note: Use GetTableData and SetTableData (only ISO-Table) for Data blockes ! <table><tr><th></th><th>bool</th><th>byte</th><th>byte[]</th><th>int</th><th>long</th><th>string</th></tr><tr><td>GetTableData</td><td></td><td></td><td>X</td><td></td><td></td><td>X</td></tr><tr><td>SetTableData</td><td></td><td></td><td>X</td><td></td><td></td><td>X</td></tr><tr><td>FindTableIndex</td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>		bool	byte	byte[]	int	long	string	GetTableData			X			X	SetTableData			X			X	FindTableIndex						
BRM-Table	ISO-Table																																
X	X																																
	bool	byte	byte[]	int	long	string																											
GetTableData			X			X																											
SetTableData			X			X																											
FindTableIndex																																	
DATA_TxDB DATA_TxDB_EPC_BANK DATA_TxDB_TID_BANK DATA_TxDB_RES_BANK <table><tr><th>BRM-Table</th><th>ISO-Table</th></tr><tr><td></td><td>X</td></tr></table>	BRM-Table	ISO-Table		X	Data blocks for send protocol Note: Use GetTableData and SetTableData (only ISO-Table) for Data blockes ! <table><tr><th></th><th>bool</th><th>byte</th><th>byte[]</th><th>int</th><th>long</th><th>string</th></tr><tr><td>GetTableData</td><td></td><td></td><td>X</td><td></td><td></td><td>X</td></tr><tr><td>SetTableData</td><td></td><td></td><td>X</td><td></td><td></td><td>X</td></tr><tr><td>FindTableIndex</td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>		bool	byte	byte[]	int	long	string	GetTableData			X			X	SetTableData			X			X	FindTableIndex						
BRM-Table	ISO-Table																																
	X																																
	bool	byte	byte[]	int	long	string																											
GetTableData			X			X																											
SetTableData			X			X																											
FindTableIndex																																	
DATA_TIMER <table><tr><th>BRM-Table</th><th>ISO-Table</th></tr><tr><td>X</td><td></td></tr></table>	BRM-Table	ISO-Table	X		Timer value from [0x22] Read Buffer. Alternativ kann auch die Methode getReaderTime von FedmBrmTableItem verwendet werden. <table><tr><th></th><th>bool</th><th>byte</th><th>byte[]</th><th>int</th><th>long</th><th>string</th></tr><tr><td>GetTableData</td><td></td><td></td><td>X</td><td>X</td><td></td><td></td></tr><tr><td>SetTableData</td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>FindTableIndex</td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>		bool	byte	byte[]	int	long	string	GetTableData			X	X			SetTableData							FindTableIndex						
BRM-Table	ISO-Table																																
X																																	
	bool	byte	byte[]	int	long	string																											
GetTableData			X	X																													
SetTableData																																	
FindTableIndex																																	
DATA_RxCB <table><tr><th>BRM-Table</th><th>ISO-Table</th></tr><tr><td></td><td>X</td></tr></table>	BRM-Table	ISO-Table		X	Configuration data block from receive protocol Note: Use GetTableData and SetTableData (only ISO-Table) for data blocks ! <table><tr><th></th><th>bool</th><th>byte</th><th>byte[]</th><th>int</th><th>long</th><th>string</th></tr><tr><td>GetTableData</td><td></td><td></td><td>X</td><td></td><td></td><td>X</td></tr><tr><td>SetTableData</td><td></td><td></td><td>X</td><td></td><td></td><td>X</td></tr><tr><td>FindTableIndex</td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>		bool	byte	byte[]	int	long	string	GetTableData			X			X	SetTableData			X			X	FindTableIndex						
BRM-Table	ISO-Table																																
	X																																
	bool	byte	byte[]	int	long	string																											
GetTableData			X			X																											
SetTableData			X			X																											
FindTableIndex																																	
DATA_TxCB <table><tr><th>BRM-Table</th><th>ISO-Table</th></tr><tr><td></td><td>X</td></tr></table>	BRM-Table	ISO-Table		X	Configuration data block for send protocol Note: Use GetTableData and SetTableData (only ISO-Table) for data blocks ! <table><tr><th></th><th>bool</th><th>byte</th><th>byte[]</th><th>int</th><th>long</th><th>String</th></tr><tr><td>GetTableData</td><td></td><td></td><td>X</td><td></td><td></td><td>X</td></tr><tr><td>SetTableData</td><td></td><td></td><td>X</td><td></td><td></td><td>X</td></tr><tr><td>FindTableIndex</td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>		bool	byte	byte[]	int	long	String	GetTableData			X			X	SetTableData			X			X	FindTableIndex						
BRM-Table	ISO-Table																																
	X																																
	bool	byte	byte[]	int	long	String																											
GetTableData			X			X																											
SetTableData			X			X																											
FindTableIndex																																	
DATA_AFI <table><tr><th>BRM-Table</th><th>ISO-Table</th></tr><tr><td></td><td>X</td></tr></table>	BRM-Table	ISO-Table		X	AFI from [0xB0] [0x2B] Get System Information <table><tr><th></th><th>bool</th><th>byte</th><th>byte[]</th><th>int</th><th>long</th><th>String</th></tr><tr><td>GetTableData</td><td></td><td>X</td><td>X</td><td>X</td><td></td><td>X</td></tr><tr><td>SetTableData</td><td></td><td>X</td><td></td><td>X</td><td></td><td>X</td></tr><tr><td>FindTableIndex</td><td></td><td>X</td><td></td><td>X</td><td></td><td></td></tr></table>		bool	byte	byte[]	int	long	String	GetTableData		X	X	X		X	SetTableData		X		X		X	FindTableIndex		X		X		
BRM-Table	ISO-Table																																
	X																																
	bool	byte	byte[]	int	long	String																											
GetTableData		X	X	X		X																											
SetTableData		X		X		X																											
FindTableIndex		X		X																													
DATA_DSFDID	DSFID from received data [0xB0] [0x01] Inventory																																

Constant	Description/Use																																
<table><tr><th>BRM-Table</th><th>ISO-Table</th></tr><tr><td></td><td>X</td></tr></table>	BRM-Table	ISO-Table		X	<table><tr><th></th><th>Bool</th><th>Byte</th><th>byte[]</th><th>int</th><th>long</th><th>String</th></tr><tr><td>GetTableData</td><td></td><td>X</td><td>X</td><td>X</td><td></td><td>X</td></tr><tr><td>SetTableData</td><td></td><td>X</td><td></td><td>X</td><td></td><td>X</td></tr><tr><td>FindTableIndex</td><td></td><td>X</td><td></td><td>X</td><td></td><td></td></tr></table>		Bool	Byte	byte[]	int	long	String	GetTableData		X	X	X		X	SetTableData		X		X		X	FindTableIndex		X		X		
BRM-Table	ISO-Table																																
	X																																
	Bool	Byte	byte[]	int	long	String																											
GetTableData		X	X	X		X																											
SetTableData		X		X		X																											
FindTableIndex		X		X																													
DATA_TRINFO	Transponder Info (only for ISO14443-4 Transponder) from received data [0xB0] [0x01] Inventory																																
<table><tr><th>BRM-Table</th><th>ISO-Table</th></tr><tr><td></td><td>X</td></tr></table>	BRM-Table	ISO-Table		X	<table><tr><th></th><th>bool</th><th>Byte</th><th>byte[]</th><th>int</th><th>long</th><th>String</th></tr><tr><td>GetTableData</td><td></td><td>X</td><td>X</td><td>X</td><td></td><td>X</td></tr><tr><td>SetTableData</td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>FindTableIndex</td><td></td><td>X</td><td></td><td>X</td><td></td><td></td></tr></table>		bool	Byte	byte[]	int	long	String	GetTableData		X	X	X		X	SetTableData							FindTableIndex		X		X		
BRM-Table	ISO-Table																																
	X																																
	bool	Byte	byte[]	int	long	String																											
GetTableData		X	X	X		X																											
SetTableData																																	
FindTableIndex		X		X																													
DATA_OPTINFO	Optional Info (nur für ISO14443A Transponder) aus Empfangsdaten [0xB0] [0x01] Inventory																																
<table><tr><th>BRM-Table</th><th>ISO-Table</th></tr><tr><td></td><td>X</td></tr></table>	BRM-Table	ISO-Table		X	<table><tr><th></th><th>bool</th><th>byte</th><th>byte[]</th><th>uint</th><th>long</th><th>string</th></tr><tr><td>GetTableData</td><td></td><td>X</td><td>X</td><td>X</td><td></td><td>X</td></tr><tr><td>SetTableData</td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>FindTableIndex</td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>		bool	byte	byte[]	uint	long	string	GetTableData		X	X	X		X	SetTableData							FindTableIndex						
BRM-Table	ISO-Table																																
	X																																
	bool	byte	byte[]	uint	long	string																											
GetTableData		X	X	X		X																											
SetTableData																																	
FindTableIndex																																	
DATA_PROTOINFO	Protocol Info (only for ISO14443B Transponder) from received data [0xB0] [0x01] Inventory																																
<table><tr><th>BRM-Table</th><th>ISO-Table</th></tr><tr><td></td><td>X</td></tr></table>	BRM-Table	ISO-Table		X	<table><tr><th></th><th>bool</th><th>byte</th><th>byte[]</th><th>uint</th><th>long</th><th>string</th></tr><tr><td>GetTableData</td><td></td><td>X</td><td>X</td><td>X</td><td></td><td>X</td></tr><tr><td>SetTableData</td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>FindTableIndex</td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>		bool	byte	byte[]	uint	long	string	GetTableData		X	X	X		X	SetTableData							FindTableIndex						
BRM-Table	ISO-Table																																
	X																																
	bool	byte	byte[]	uint	long	string																											
GetTableData		X	X	X		X																											
SetTableData																																	
FindTableIndex																																	
DATA_FSCI	Max. Frame Size (only for ISO14443-4 Transponder) from received data [0xB2] [0x2B] ISO14443-4 Transponder Info																																
<table><tr><th>BRM-Table</th><th>ISO-Table</th></tr><tr><td></td><td>X</td></tr></table>	BRM-Table	ISO-Table		X	<table><tr><th></th><th>bool</th><th>byte</th><th>byte[]</th><th>uint</th><th>long</th><th>string</th></tr><tr><td>GetTableData</td><td></td><td>X</td><td>X</td><td>X</td><td></td><td>X</td></tr><tr><td>SetTableData</td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>FindTableIndex</td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>		bool	byte	byte[]	uint	long	string	GetTableData		X	X	X		X	SetTableData							FindTableIndex						
BRM-Table	ISO-Table																																
	X																																
	bool	byte	byte[]	uint	long	string																											
GetTableData		X	X	X		X																											
SetTableData																																	
FindTableIndex																																	
DATA_FWI	Frame Waiting Time (only for ISO14443-4 Transponder) from received data [0xB2] [0x2B] ISO14443-4 Transponder Info																																
<table><tr><th>BRM-Table</th><th>ISO-Table</th></tr><tr><td></td><td>X</td></tr></table>	BRM-Table	ISO-Table		X	<table><tr><th></th><th>bool</th><th>byte</th><th>byte[]</th><th>uint</th><th>long</th><th>string</th></tr><tr><td>GetTableData</td><td></td><td>X</td><td>X</td><td>X</td><td></td><td>X</td></tr><tr><td>SetTableData</td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>FindTableIndex</td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>		bool	byte	byte[]	uint	long	string	GetTableData		X	X	X		X	SetTableData							FindTableIndex						
BRM-Table	ISO-Table																																
	X																																
	bool	byte	byte[]	uint	long	string																											
GetTableData		X	X	X		X																											
SetTableData																																	
FindTableIndex																																	
DATA_DSI	Devisor Send Integer (only for ISO14443-4 Transponder) from received data [0xB2] [0x2B] ISO14443-4 Transponder Info																																
<table><tr><th>BRM-Table</th><th>ISO-Table</th></tr><tr><td></td><td>X</td></tr></table>	BRM-Table	ISO-Table		X	<table><tr><th></th><th>bool</th><th>byte</th><th>byte[]</th><th>uint</th><th>long</th><th>string</th></tr><tr><td>GetTableData</td><td></td><td>X</td><td>X</td><td>X</td><td></td><td>X</td></tr><tr><td>SetTableData</td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>FindTableIndex</td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>		bool	byte	byte[]	uint	long	string	GetTableData		X	X	X		X	SetTableData							FindTableIndex						
BRM-Table	ISO-Table																																
	X																																
	bool	byte	byte[]	uint	long	string																											
GetTableData		X	X	X		X																											
SetTableData																																	
FindTableIndex																																	
DATA_DRI	Devisor Receive Integer (only for ISO14443-4 Transponder) from received																																

Constant	Description/Use																																
<table><tr><th>BRM-Table</th><th>ISO-Table</th></tr><tr><td></td><td>X</td></tr></table>	BRM-Table	ISO-Table		X	<div>data [0xB2] [0x2B] ISO14443-4 Transponder Info</div> <table><tr><th></th><th>bool</th><th>byte</th><th>byte[]</th><th>uint</th><th>long</th><th>string</th></tr><tr><td>GetTableData</td><td></td><td>X</td><td>X</td><td>X</td><td></td><td>X</td></tr><tr><td>SetTableData</td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>FindTableIndex</td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>		bool	byte	byte[]	uint	long	string	GetTableData		X	X	X		X	SetTableData							FindTableIndex						
BRM-Table	ISO-Table																																
	X																																
	bool	byte	byte[]	uint	long	string																											
GetTableData		X	X	X		X																											
SetTableData																																	
FindTableIndex																																	
<div>DATA_NAD</div> <table><tr><th>BRM-Table</th><th>ISO-Table</th></tr><tr><td></td><td>X</td></tr></table>	BRM-Table	ISO-Table		X	<div>Node Address (only for ISO14443-4 Transponder) from received data [0xB2] [0x2B] ISO14443-4 Transponder Info</div> <table><tr><th></th><th>bool</th><th>byte</th><th>byte[]</th><th>uint</th><th>long</th><th>string</th></tr><tr><td>GetTableData</td><td></td><td>X</td><td>X</td><td>X</td><td></td><td>X</td></tr><tr><td>SetTableData</td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>FindTableIndex</td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>		bool	byte	byte[]	uint	long	string	GetTableData		X	X	X		X	SetTableData							FindTableIndex						
BRM-Table	ISO-Table																																
	X																																
	bool	byte	byte[]	uint	long	string																											
GetTableData		X	X	X		X																											
SetTableData																																	
FindTableIndex																																	
<div>DATA_CID</div> <table><tr><th>BRM-Table</th><th>ISO-Table</th></tr><tr><td></td><td>X</td></tr></table>	BRM-Table	ISO-Table		X	<div>Card Identifier (only for ISO14443-4 Transponder) from received data [0xB2] [0x2B] ISO14443-4 Transponder Info</div> <table><tr><th></th><th>bool</th><th>byte</th><th>byte[]</th><th>uint</th><th>long</th><th>string</th></tr><tr><td>GetTableData</td><td></td><td>X</td><td>X</td><td>X</td><td></td><td>X</td></tr><tr><td>SetTableData</td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>FindTableIndex</td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>		bool	byte	byte[]	uint	long	string	GetTableData		X	X	X		X	SetTableData							FindTableIndex						
BRM-Table	ISO-Table																																
	X																																
	bool	byte	byte[]	uint	long	string																											
GetTableData		X	X	X		X																											
SetTableData																																	
FindTableIndex																																	
<div>DATA_SEC_STATUS</div> <table><tr><th>BRM-Table</th><th>ISO-Table</th></tr><tr><td></td><td>X</td></tr></table>	BRM-Table	ISO-Table		X	<div>Security Status from received data [0xB0] [0x23] Read Multiple Blocks</div> <div>Note: Use GetTableData and SetTableData (only ISO-Table) for data blocks !</div> <table><tr><th></th><th>bool</th><th>byte</th><th>byte[]</th><th>int</th><th>long</th><th>String</th></tr><tr><td>GetTableData</td><td></td><td></td><td>X</td><td></td><td></td><td>X</td></tr><tr><td>SetTableData</td><td></td><td></td><td>X</td><td></td><td></td><td>X</td></tr><tr><td>FindTableIndex</td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>		bool	byte	byte[]	int	long	String	GetTableData			X			X	SetTableData			X			X	FindTableIndex						
BRM-Table	ISO-Table																																
	X																																
	bool	byte	byte[]	int	long	String																											
GetTableData			X			X																											
SetTableData			X			X																											
FindTableIndex																																	
<div>DATA_BLOCK_SIZE</div> <table><tr><th>BRM-Table</th><th>ISO-Table</th></tr><tr><td></td><td>X</td></tr></table>	BRM-Table	ISO-Table		X	<div>Block size from received data [0xB0] [0x23] Read Multiple Blocks</div> <table><tr><th></th><th>bool</th><th>byte</th><th>byte[]</th><th>int</th><th>long</th><th>String</th></tr><tr><td>GetTableData</td><td></td><td>X</td><td>X</td><td>X</td><td></td><td></td></tr><tr><td>SetTableData</td><td></td><td>X</td><td></td><td>X</td><td></td><td>X</td></tr><tr><td>FindTableIndex</td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>		bool	byte	byte[]	int	long	String	GetTableData		X	X	X			SetTableData		X		X		X	FindTableIndex						
BRM-Table	ISO-Table																																
	X																																
	bool	byte	byte[]	int	long	String																											
GetTableData		X	X	X																													
SetTableData		X		X		X																											
FindTableIndex																																	
<div>DATA_MEM_SIZE</div> <table><tr><th>BRM-Table</th><th>ISO-Table</th></tr><tr><td></td><td>X</td></tr></table>	BRM-Table	ISO-Table		X	<div>Memory size from [0xB0] [0x2B] Get System Information</div> <table><tr><th></th><th>bool</th><th>byte</th><th>byte[]</th><th>int</th><th>long</th><th>string</th></tr><tr><td>GetTableData</td><td></td><td></td><td>X</td><td></td><td></td><td>X</td></tr><tr><td>SetTableData</td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>FindTableIndex</td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>		bool	byte	byte[]	int	long	string	GetTableData			X			X	SetTableData							FindTableIndex						
BRM-Table	ISO-Table																																
	X																																
	bool	byte	byte[]	int	long	string																											
GetTableData			X			X																											
SetTableData																																	
FindTableIndex																																	
<div>DATA_IC_REF</div> <table><tr><th>BRM-Table</th><th>ISO-Table</th></tr><tr><td></td><td>X</td></tr></table>	BRM-Table	ISO-Table		X	<div>IC-Reference from [0xB0] [0x2B] Get System Information</div> <table><tr><th></th><th>bool</th><th>byte</th><th>byte[]</th><th>int</th><th>long</th><th>string</th></tr><tr><td>GetTableData</td><td></td><td>X</td><td>X</td><td>X</td><td></td><td>X</td></tr><tr><td>SetTableData</td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>FindTableIndex</td><td></td><td>X</td><td></td><td>X</td><td></td><td></td></tr></table>		bool	byte	byte[]	int	long	string	GetTableData		X	X	X		X	SetTableData							FindTableIndex		X		X		
BRM-Table	ISO-Table																																
	X																																
	bool	byte	byte[]	int	long	string																											
GetTableData		X	X	X		X																											
SetTableData																																	
FindTableIndex		X		X																													
<div>DATA_DB_ADR</div>	<div>Data block address from received data[] [0x22] Read Buffer</div> <table><tr><th></th><th>bool</th><th>byte</th><th>byte[]</th><th>int</th><th>long</th><th>string</th></tr><tr><td>GetTableData</td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>SetTableData</td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>FindTableIndex</td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>		bool	byte	byte[]	int	long	string	GetTableData							SetTableData							FindTableIndex										
	bool	byte	byte[]	int	long	string																											
GetTableData																																	
SetTableData																																	
FindTableIndex																																	

Constant		Description/Use						
BRM-Table	ISO-Table	GetTableData		X	X	X		X
X		SetTableData						
		FindTableIndex						
DATA_DBN		Number of data blocks from receive data [0x22] Read Buffer						
BRM-Table	ISO-Table		bool	byte	byte[]	int	long	string
X		GetTableData		X	X		X	X
		SetTableData						
		FindTableIndex						
DATA_IS_BLOCK_SIZE_SET		Flag, whether block size was set with [0xB0] [0x23] Read Multiple Blocks						
BRM-Table	ISO-Table		bool	byte	byte[]	int	long	string
	X	GetTableData	X	X	X	X		
		SetTableData	X	X		X		
		FindTableIndex	X					
DATA_IS_SELECTED		Flag, whether transponder is in selected mode. Is set with [0xB0][0x25] Select						
BRM-Table	ISO-Table		bool	byte	byte[]	int	long	string
	X	GetTableData	X	X	X	X		
		SetTableData	X	X		X		
		FindTableIndex	X					
DATA_IS_ISO14443_4_INFO		Flag, whether transponder info data are read with [0xB2] [0x2B] ISO14443-4 Transponder Info						
BRM-Table	ISO-Table		bool	byte	byte[]	uint	long	string
	X	GetTableData	X	X	X	X		
		SetTableData	X	X		X		
		FindTableIndex						
DATA_EPC		EPC; (EPC = Electronic Product Code) from receive data [0xB0] [0x01] Inventory. The EPC is a string in the format: "xx.xxxxxx.xxxxxx.xxxxxx" (Header.DomainManager.ObjectClass.Seriennummer).						
BRM-Table	ISO-Table		bool	byte	byte[]	int	long	string
X	X	GetTableData			X			X
		SetTableData						
		FindTableIndex						X

Constant	Description/Use																																		
<div>DATA_EPC_TYPE</div> <div><table><tr><th>BRM-Table</th><th>ISO-Table</th></tr><tr><td>X</td><td>X</td></tr></table></div>	BRM-Table	ISO-Table	X	X	<div>EPC-Typ; (EPC = Electronic Product Code) from received data [0xB0] [0x01] Inventory. The EPC-Type is extracted from the field EPC-Header.</div> <div><table><tr><th></th><th>bool</th><th>byte</th><th>byte[]</th><th>int</th><th>Long</th><th>string</th></tr><tr><td>GetTableData</td><td></td><td>X</td><td>X</td><td>X</td><td></td><td></td></tr><tr><td>SetTableData</td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>FindTableIndex</td><td></td><td>X</td><td></td><td>X</td><td></td><td></td></tr></table></div>		bool	byte	byte[]	int	Long	string	GetTableData		X	X	X			SetTableData							FindTableIndex		X		X				
BRM-Table	ISO-Table																																		
X	X																																		
	bool	byte	byte[]	int	Long	string																													
GetTableData		X	X	X																															
SetTableData																																			
FindTableIndex		X		X																															
<div>DATA_EPC_HEADER</div> <div><table><tr><th>BRM-Table</th><th>ISO-Table</th></tr><tr><td>X</td><td>X</td></tr></table></div>	BRM-Table	ISO-Table	X	X	<div>Field EPC Header; (EPC = Electronic Product Code) from received data [0xB0] [0x01] Inventory</div> <div><table><tr><th></th><th>bool</th><th>byte</th><th>byte[]</th><th>int</th><th>long</th><th>String</th></tr><tr><td>GetTableData</td><td></td><td></td><td></td><td>X</td><td></td><td>X</td></tr><tr><td>SetTableData</td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>FindTableIndex</td><td></td><td></td><td></td><td>X</td><td></td><td></td></tr></table></div>		bool	byte	byte[]	int	long	String	GetTableData				X		X	SetTableData							FindTableIndex				X				
BRM-Table	ISO-Table																																		
X	X																																		
	bool	byte	byte[]	int	long	String																													
GetTableData				X		X																													
SetTableData																																			
FindTableIndex				X																															
<div>DATA_EPC_DOMAIN</div> <div><table><tr><th>BRM-Table</th><th>ISO-Table</th></tr><tr><td>X</td><td>X</td></tr></table></div>	BRM-Table	ISO-Table	X	X	<div>Field EPC-DomainManager; (EPC = Electronic Product Code) from received data [0xB0] [0x01] Inventory</div> <div><table><tr><th></th><th>bool</th><th>byte</th><th>byte[]</th><th>int</th><th>long</th><th>string</th></tr><tr><td>GetTableData</td><td></td><td></td><td></td><td></td><td>X</td><td>X</td></tr><tr><td>SetTableData</td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>FindTableIndex</td><td></td><td></td><td></td><td></td><td>X</td><td>X</td></tr></table></div>		bool	byte	byte[]	int	long	string	GetTableData					X	X	SetTableData							FindTableIndex					X	X		
BRM-Table	ISO-Table																																		
X	X																																		
	bool	byte	byte[]	int	long	string																													
GetTableData					X	X																													
SetTableData																																			
FindTableIndex					X	X																													
<div>DATA_EPC_OBJECT</div> <div><table><tr><th>BRM-Table</th><th>ISO-Table</th></tr><tr><td>X</td><td>X</td></tr></table></div>	BRM-Table	ISO-Table	X	X	<div>Field EPC-ObjectClass; (EPC = Electronic Product Code) from received data [0xB0] [0x01] Inventory</div> <div><table><tr><th></th><th>bool</th><th>byte</th><th>byte[]</th><th>int</th><th>long</th><th>String</th></tr><tr><td>GetTableData</td><td></td><td></td><td></td><td></td><td>X</td><td>X</td></tr><tr><td>SetTableData</td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>FindTableIndex</td><td></td><td></td><td></td><td></td><td>X</td><td>X</td></tr></table></div>		bool	byte	byte[]	int	long	String	GetTableData					X	X	SetTableData							FindTableIndex					X	X		
BRM-Table	ISO-Table																																		
X	X																																		
	bool	byte	byte[]	int	long	String																													
GetTableData					X	X																													
SetTableData																																			
FindTableIndex					X	X																													
<div>DATA_EPC_SNR</div> <div><table><tr><th>BRM-Table</th><th>ISO-Table</th></tr><tr><td>X</td><td>X</td></tr></table></div>	BRM-Table	ISO-Table	X	X	<div>Feld EPC-Seriennummer; (EPC = Electronic Product Code) from received data [0xB0] [0x01] Inventory</div> <div><table><tr><th></th><th>bool</th><th>byte</th><th>byte[]</th><th>int</th><th>long</th><th>string</th></tr><tr><td>GetTableData</td><td></td><td></td><td></td><td></td><td>X</td><td>X</td></tr><tr><td>SetTableData</td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>FindTableIndex</td><td></td><td></td><td></td><td></td><td>X</td><td>X</td></tr></table></div>		bool	byte	byte[]	int	long	string	GetTableData					X	X	SetTableData							FindTableIndex					X	X		
BRM-Table	ISO-Table																																		
X	X																																		
	bool	byte	byte[]	int	long	string																													
GetTableData					X	X																													
SetTableData																																			
FindTableIndex					X	X																													
<div>DATA_DATE</div> <div><table><tr><th>BRM-Table</th><th>ISO-Table</th></tr><tr><td>X</td><td></td></tr></table></div>	BRM-Table	ISO-Table	X		<div>Date field from receive protocol [0x22] Read Buffer</div> <div><table><tr><th></th><th>FeliscReaderTime</th><th></th><th></th><th></th><th></th></tr><tr><td>GetReaderTime</td><td>X</td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td></tr></table></div>		FeliscReaderTime					GetReaderTime	X																						
BRM-Table	ISO-Table																																		
X																																			
	FeliscReaderTime																																		
GetReaderTime	X																																		

Constant	Description/Use																																
<div>DATA_ANT_NR</div> <table><tr><th>BRM-Table</th><th>ISO-Table</th></tr><tr><td>X</td><td></td></tr></table>	BRM-Table	ISO-Table	X		<div>Antenna number from receive protocol [0x22] Read Buffer</div> <table><tr><th></th><th>bool</th><th>byte</th><th>byte[]</th><th>int</th><th>long</th><th>string</th></tr><tr><td>GetTableData</td><td></td><td>X</td><td>X</td><td>X</td><td></td><td>X</td></tr><tr><td>SetTableData</td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>FindTableIndex</td><td></td><td>X</td><td></td><td>X</td><td></td><td></td></tr></table>		bool	byte	byte[]	int	long	string	GetTableData		X	X	X		X	SetTableData							FindTableIndex		X		X		
BRM-Table	ISO-Table																																
X																																	
	bool	byte	byte[]	int	long	string																											
GetTableData		X	X	X		X																											
SetTableData																																	
FindTableIndex		X		X																													
<div>FEDM_ISC_DATA_INPUT</div> <table><tr><th>BRM-Table</th><th>ISO-Table</th></tr><tr><td>X</td><td></td></tr></table>	BRM-Table	ISO-Table	X		<div>Input byte from receive protocol [0x22] Read Buffer</div> <table><tr><th></th><th>bool</th><th>byte</th><th>byte[]</th><th>uint</th><th>long</th><th>string</th></tr><tr><td>GetTableData</td><td></td><td>X</td><td></td><td>X</td><td>X</td><td>X</td></tr><tr><td>SetTableData</td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>FindTableIndex</td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>		bool	byte	byte[]	uint	long	string	GetTableData		X		X	X	X	SetTableData							FindTableIndex						
BRM-Table	ISO-Table																																
X																																	
	bool	byte	byte[]	uint	long	string																											
GetTableData		X		X	X	X																											
SetTableData																																	
FindTableIndex																																	
<div>FEDM_ISC_DATA_STATE</div> <table><tr><th>BRM-Table</th><th>ISO-Table</th></tr><tr><td>X</td><td></td></tr></table>	BRM-Table	ISO-Table	X		<div>Status byte from receive protocol [0x22] Read Buffer</div> <table><tr><th></th><th>bool</th><th>byte</th><th>byte[]</th><th>uint</th><th>long</th><th>string</th></tr><tr><td>GetTableData</td><td></td><td>X</td><td></td><td>X</td><td>X</td><td>X</td></tr><tr><td>SetTableData</td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>FindTableIndex</td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>		bool	byte	byte[]	uint	long	string	GetTableData		X		X	X	X	SetTableData							FindTableIndex						
BRM-Table	ISO-Table																																
X																																	
	bool	byte	byte[]	uint	long	string																											
GetTableData		X		X	X	X																											
SetTableData																																	
FindTableIndex																																	
<div>FEDM_ISC_DATA_MAC_ADR</div> <table><tr><th>BRM-Table</th><th>ISO-Table</th></tr><tr><td>X</td><td></td></tr></table>	BRM-Table	ISO-Table	X		<div>Status byte from receive protocol [0x22] Read Buffer</div> <table><tr><th></th><th>bool</th><th>byte</th><th>byte[]</th><th>uint</th><th>long</th><th>string</th></tr><tr><td>GetTableData</td><td></td><td></td><td>X</td><td></td><td></td><td>X</td></tr><tr><td>SetTableData</td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>FindTableIndex</td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table>		bool	byte	byte[]	uint	long	string	GetTableData			X			X	SetTableData							FindTableIndex						
BRM-Table	ISO-Table																																
X																																	
	bool	byte	byte[]	uint	long	string																											
GetTableData			X			X																											
SetTableData																																	
FindTableIndex																																	

10.6. Revision history

V4.08.03

- Updated Reader Configuration namespaces in ReaderConfig
- New methods in Reader class FedmlscReader:
 - CancelTclApdu
 - GetIsoErrorCode

V4.08.02

- - Support LRU1002: Phase Angel feature
- - Improvements TagHandler MifarePlus SL3
- - Improvements TagHandler UHF EPC Class 1 Gen 2 - write EPC with len=0

V4.07.00

- Update of namespaces and access constants for reader configuration
- Support for new Reader type : ID CPR74
- Support for improved Reader type ID ISC.LRU1002
- Support for UHF transponder type UCODE DNA
- Support for ISO 14443 transponder type NXP Ultralight EV1
- Bugfix in [0xC3] FlexSoftCrypto command GetValue() of TagHandler class Mifare DESFire

V4.06.16

- Discontinued support for .NET Framework 2.0.
- Discontinued support for Windows XP and Windows CE5.
- Bugfix for USB event notification
- Bugfix for Timeout parameter in StartAsyncTask

V4.06.15

- Sample projects build with IDE Version Visual Studio 2012.
- Support for ASP.NET projects.
- Integration of libraries FedmlscCore and FedmlscMyAxxess inside the OBIDISC4NETnative library.
- Extension of MyAxxess feature – Online Access Request (ID MAX50 reader).
- No longer necessary - Installation of Microsoft redistributable packages (excluded Win10)

- Bugfix for “Asynchronous Inventory” feature.
- Update of namespaces and access constants for reader configuration.

V4.06.06

- TagHandler support for ISO 14443 Transponder with 10 byte UID
- Bugfix for EPC Class1 Gen2 Transponder with Extended PC
- ISO 15693: [0x2C] Get Multiple Block Security Status with extended addressed mode: bugfix for received data above address 255
- Update of namespaces and access constants for reader configuration

V4.06.01

- Support for new Readers: : **ID ISC.PRH200, ID ISC.LRU1002, ID myAXXESS onTop**
- New TagHandler class:
 - FedmlscTagHandler_ISO15693_STM_LRIS64K
 - FedmlscTagHandler_ISO15693_STM_M24LR64R
 - FedmlscTagHandler_EPC_Class1_Gen2_IDS_SL900A
- New method Convert_EPC_C1_G2_TagHandler in the Reader class **FedmlscReader**
- Update of namespaces and access constants for reader configuration

V4.05.00

- Bugfix in Reader class **FedmlscReader**: **AbandonedMutexException** fixed in the methods TagInventory and TagSelect.
- New method GetDependentLibVersions in the Reader class **FedmlscReader**
- Modifications for method **StartAsyncTask** of Reader class **FedmlscReader**:
 - a) While initializing the asynchronous Task for Reader's Notification-Mode, the Listener Port must be unused in the system. Otherwise, the error code -4086 is returned.
 - b) The Listener Port for Reader's Notification-Mode accepts only one connection at the same time. All additional connections will be rejected.
- **FedmlscTagHandler_ISO15693_NXP_ICODE_SLI_L**: new method PasswordProtectAFI
- Update of namespaces and access constants for reader configuration

V4.04.01

- Bugfix in TagHandler class **FedmlscTagHandler_EPC_Class1_Gen2**:
Calculation of password length in methods Kill and Lock fixed.

- Update of namespaces and access constants for reader configuration

V4.04.00

- Support for 64-Bit .NET-Framework, beginning with Framework V4.0
- New namespace **OBID.ReaderCommand** containing all Command Parameters, ordered by groups (this collection does not replace the constants in **FedmlscReaderID**, but it is recommended as the better and more intuitive applicable alternative)
- New TagHandler class: **FedmlscTagHandler_ISO18000_3M3**
- New methods in TagHandler class **FedmlscTagHandler_EPC_Class1_Gen2**:
 - GetTagModelNumber
 - GetMaskDesignerID
 - GetMaskDesignerName
 - IsUidWithTid
 - IsExtendedPC_W1
 - GetExtendedProtocolControlW1
- Class **FedmlscReader**: Command [0x6E]: support for Mode 0x21
- Class **FedmBrmTableItem**: New element: class1Gen2XPC_W1 (Extended PC Word 1)
- Class **FedmlsoTableItem**: New element: class1Gen2XPC_W1 (Extended PC Word 1)
- New elements in structure **FedmlscReaderInfo**: version number of embedded ACC application in *AccEmbAppSwVer* and *AccEmbAppDevVer*
- Bugfix for Command [0x77] Get People Counter Values: false value in counter2 corrected.
- Update of namespaces and access constants for reader configuration

V4.03.00

- Support for new Reader: **ID CPR46.xx**
- Support for new Transponder: Innovatron (ISO 14443B') und ASK CTx
- New TagHandler classes:
 1. **FedmlscTagHandler_ISO14443_Innovatron**
 2. **FedmlscTagHandler_ISO14443_3_ASK_CTx**
- Class **FedmlscReader**:
 1. Support for Gate People Counter in Notification Mode
 2. New overloaded method *SetTableSize* to adjust additionally the buffers for Transponder data.
 3. New, overloaded methods *ConnectCOMM* and *ConnectUSB* for secured data transmission.
- Class **FedmBrmTableItem**:
 - Almost all data elements are now public for direct access
 - Support for direction information in combination with Gate People Counter

- New table elements: IDDT, AFI and DSFID
- Class **FedmlsoTableItem**: Almost all data elements are now public for direct access
- Class **FeliscReaderTime**: Format of date modified for compliance with ISO 8601
- Interface **FedmTaskListener**: new method *OnNewPeopleCounterEvent*
- Bugfix in *ReadCompleteBank* of **FedmlscTagHandler_EPC_Class1_Gen2**: repeat of data after multiple of 166 bytes
- Update of namespaces and access constants for reader configuration
- Rename of Namespaces:

Old	New
OperatingMode.xxMode.DataSource. MifareAppID	OperatingMode.xxMode.DataSource. Mifare.Classic.AppID
OperatingMode.xxMode.DataSource. MifareKeyAddress	OperatingMode.xxMode.DataSource. Mifare.Classic.KeyAddress
OperatingMode.xxMode.DataSource. MifareKeyType	OperatingMode.xxMode.DataSource. Mifare.Classic.KeyType

Note: xxMode stands for NotificationMode or ScanMode

V4.02.00

- First release for .NET Framework 4.0
- Improved thread safeness
- Support for new Reader: **ID ISC.LR1002**
- Class **FedmlscReader**:
 1. Rename of the method *GetNonAddressedTagHandler* in *CreateNonAddressedTagHandler*.
 2. Add of *Dispose* method
- **TagHandler class for ISO 14443-4 Mifare DESFire with FlexSoft- und SAM-Crypto**: Bugfix in the method *SetConfiguration* with values of 1 and 2 in *Parameter* option
- **ISO 14443-3 bzw. -4 Transponder**: Optimized *Select-Algorithm* in the method *TagSelect*.
- **EPC Class 1 Gen 2**:
 1. Support for non-addressed mode.
 2. When *UID = EPC + TID* is configured: return of an errorcode (-158) before executing of an tag command in addressed mode, if *UID* contains no *TID*.
- **TagHandler class for EPC Class 1 Gen 2**:
 1. *ISO-Errorcode* is a new class member.
 2. Method *GetTidOfUid* returns *TID* even if length of *EPC* is zero.
- Update of namespaces and access constants for reader configuration
- Dynamic binding to Log-Manager

V4.00.07

- Update of namespaces and access constants for reader configuration
- TagHandler class for EPC Class1 Gen2: new method Lock with simplified parameter list

V4.00.03

- Bugfix in TagHandler class for EPC Class1 Gen2 in method ReadCompleteBank

V4.00.02

- Update of namespaces and access constants for reader configuration
- Keep-Alive option for Notification Tasks enabled by default in helper class FedmTaskOption
- Check for double UIDs in the method FEDM_ISCReaderModule::TagInventory
- TagHandler for EPC Class1 Gen2: new method ReadCompleteBank
- Bugfix in the method FEDM_ISCReaderModule::ReadCompleteConfiguration for ID ISC.LR2500-A and ID ISC.LRU3000
- Only for Windows: Dependency from MFC and CRT libraries according MS11-025¹²

V4.00.01

- Bugfix in TagHandler-Class for EPC Class1 Gen2 in method ReadMultipleBlock: EPC-Bank, TID-Bank and RES-Bank are now supported

V4.00.00

- Update of namespaces and access constants for reader configuration
- Support for new Reader: **ID ISC.LR2500-A**
- Support for UIDs up to 96 Bytes
- Support for UHF-Configuration UID = EPC + TID
- The organization of the Reader configuration for **ID ISC.LRU3000** above CFG63 is modified with firmware version from V2.0.0 and no longer compatible with the previous version. This SDK version adds the necessary adaptations and is therefore no longer compatible for firmware versions less than V2.0.0. The Reader classes do not check of compatibility. This must be done on application-side.

The table below summarizes the compatibilities:

LRU3000-Firmware	use SDK-Version	use ISOStart-Version	XML-Configuration file
------------------	-----------------	----------------------	------------------------

¹² Microsoft Security Bulletin Article-ID: 2538218 from Juni 14, 2011

LRU3000-Firmware	use SDK-Version	use ISOStart-Version	XML-Configuration file
< 2.00.00	<= 3.03.01	<= 8.03.02	must be created with ISOStart <= 8.03.02
>= 2.00.00	>= 4.00.00	>= 9.00.00	must be created with ISOStart >= 9.00.00

- The shared use of a TCP/IP connection from different reader objects is no longer supported. ConnectTCP returns with error code -157, if another reader object tries to connect to a Reader with the same IP-Address and Port, which is still connected
- The method Disconnect of Reader class **FedmlscReader** has a new signature: the return type void is changed into int to provide to return a positive value, if in case of a TCP/IP connection the closing was not successful. The positive return value represents the last status of the connection. It is recommended to view each code line, which call this method.
- New method in the Reader class **FEDM_ISCReaderModule**: *GetTcpConnectionState*
- Support for [0x74] Input Event with Notification-Mode for the Reader **ID CPR50** and **ID MAX50**
- TagHandler-Class for EPC Class1 Gen2 with new methods: *GetProtocolControl*, *GetEpcOfUid*, *GetTidOfUid*
- The class **FedmTaskOption** is extended with new parameters for the Keep-Alive option inside the Notification-Task. The KeepAlive option is enabled by default. If the Keep-Alive option is activated (recommended), then the listener socket is closed automatically after a break of the network cable or after loss of power and is recovered again. This ensures the reliability of the network connection.

V3.03.01

- Update of namespaces and access constants for reader configuration
- Support for new Reader: **ID ISC.MRU102**
- Error corrections in TagHandler classes
- First release for Windows CE

V3.03.00

- Update of namespaces and access constants for reader configuration
- Support for new Reader: **ID ISC.LR2500-B**, **ID ISC.MR102**, **ID CPR30.xx** und **ID ISC.CPR52.xx**

V3.02.09

- Transponder classes (TagHandler) with efficient API for standardized Transponder (ISO 15693, ISO 14443, EPC Class 1 Gen 2) or Transponder with manufacturer specific commands.
- Namespace OBID.TagHandler contains all TagHandler classes

- New methods in the Reader class **FedmlscReader**:
 1. Synchronous SAM-Command (SendSAMCommand)
 2. TagInventory
 3. TagSelect
 4. GetTagList
 5. GetTagHandler
 6. GetNonAddressedTagHandler
 7. GetSelectedTagHandler

V3.02.04

- New method **SetTableSize** in Reader-Class **FedmlscReader** to customize the size of the buffer for transponder data in the table für Host-Commands (ISO_TABLE). With this method the limit of 256 data blocks can be set with a new value (e.g. 2048).

V3.02.01

- Support for HF-Gates with People Counter ID **ISC.ANT1690/600-GPC** and ID **ISC.ANT1700/740-GPC**
- New reader configuration parameters in the package **OBID.ReaderConfig**.
- Support for RSSI measurements in all Reader Modes for Reader ID **ISC.LRU3000**

V3.01.06

- Support for new Reader: ID **ISC.LRU3000**, ID **CPR44.xx**, ID **MAX50.xx**
- New reader configuration parameters in the package **OBID.ReaderConfig**.
- New option for encrypted data transmission by use of openSSL library in the version 0.9.8l (s.

[5.3.3.Secured data transmission with encryption\).](#)

- Extension of the class **FedmlscReaderInfo** to support new features with command [0x66] Reader Info.

V3.00.13

- Bugfix for Garbage Collector in class **FedmlscReader**.
- New method **GetReport()** in class **FedmlscReaderInfo**.

V3.00.07

- Support for new reader: **ID CPR50.xx**.
- Support for transponder type NXP MIFARE DESFire.
- New high-level methods in **FedmlscReaderModule** for OBID®*classic-pro* Reader
 - **SendSAMComand**
 - **SendTclApdu**
 - **SendTclPing**
 - **SendTclDeselect**
 - **SendCommandQueue**
 - **AddCommandToQueue**
- New high-level methods in **FedmlscReaderModule** for all OBID® Reader
 - **TransferReaderCfgToXmlFile**
 - **TransferXmlFileToReaderCfg**
- New class **FedmCprApdu** for asynchronous ISO14443-4 T=CL commands with OBID®*classic-pro* Reader.
- New class **FedmCprCommandQueue** for asynchronous execution of [0xBC] Command Queue with OBID®*classic-pro* Reader.
- Modifications in the Reader class **FedmlscReader**:
 - The methods **ConnectUSB** and **ConnectTCP** execute internally a **ReadReaderInfo** to query important information from the connected Reader.
 - The method **ConnectCOMM** opens a serial port and can optional, but recommended, execute internally a **FindBaudrate** to determine the port parameters and, if the Reader is detected successfully, a **ReadReaderInfo** to query important information from the connected Reader.
 - The methode **SendProtocol(0x72)** use internally modified definitions of the constants **FEDM_ISC_TMP_0x72_OUT_TYPE_1...FEDM_ISC_TMP_0x72_OUT_TYPE_8**: Up to the previous release they adresses one bit. Now they addresses three bits. Thus, the OUT-TYPE 'Relay' must be set to 0x04 instead of 0x01 ([8.1. Basic commands](#)). This is applied to all reader types which supports the command [0x72] Set Output.

V3.00.00

- Support for new reader: **ID ISC.MRU200**, **ID ISC.PRHD102** and **ID CPR40.xx**.
- The following older reader types are no longer supported: ID ISC.M01 and ID ISC.LR100.
- Support for UHF-Multiplexer **ID ISC.ANT.UMUX**.
- Support for transponder type EPC Class1 Gen2 HF.
- Automatic detection of version conflicts with dependent library files.
- New high-level methods in **FedmlscReaderModule**
 - ApplyConfiguration
 - ReadCompleteConfiguration
 - WriteCompleteConfiguration
 - ResetCompleteConfiguration
 - ReadReaderInfo
- New class **FedmlscReaderInfo** collecting important information from the connected reader.
- Collecting of all access constants for reader configuration in the namespace OBID.ReaderConfig improves the clearness. Thus, the structures FedmlscReaderID_MR200, FedmlscReaderID_LR200, FedmlscReaderID_LR2000, FedmlscReaderID_LRU1000, FedmlscReaderID_LRU2000 as well as the access constants for OBID i-scan® Short- and Mid-Range reader and OBID® classic-pro reader in the structur FedmlscReaderID are removed.
- New overloaded methods Get/SetConfigPara in class **FedmlscReader** for modifying reader configuration parameters in the namespace OBID.ReaderConfig.
- Writing of reader configuration is only possible for previous read configuration blocks except, if the reader configuration is load by a XML file.

V2.05.01

- Modified licence agreement
- Support for the UHF-Reader ID ISC.LRU2000. The interface **FedmlscReaderID_LRU2000** contains additional constants for the configuration, which differ from the UHF-Reader ID ISC.LRU1000.
- Extensions for the UHF-Reader ID ISC.LRU1000 in the Interface **FedmlscReaderID_LRU1000**.
- New methods in the reader class **FedmlscReader** supporting asynchronous tasks (only for .NET 2.0).
- New interface **FedmTaskListener** (only for .NET 2.0).
- New property class **FedmTaskOption** (only for .NET 2.0).

V2.04.00

- Support for .NET 2.0

- Sign of the Assembly files OBIDISC4NETnative.DLL and OBIDISC4NET.DLL for .NET 2.0 with a strong name
- New common constants for the UHF-Reader LRU1000:

Constant	Comment
FEDM_ISC_LRU1000_EE_SELmask_EPC_CL1_GEN1_MASK_LGT	Constants for Selection Mask in the reader configuration for the transponder type EPC Class 1 Gen 1
FEDM_ISC_LRU1000_EE_SELmask_EPC_CL1_GEN1_MASK_START_PTR	
FEDM_ISC_LRU1000_EE_SELmask_EPC_CL1_GEN1_MASK	
FEDM_ISC_LRU1000_EE_SELmask_EPC_CL1_GEN2_MASK_LGT	Constants for Selection Mask in the reader configuration for the transponder type EPC Class 1 Gen 2
FEDM_ISC_LRU1000_EE_SELmask_EPC_CL1_GEN2_MASK_MODE	
FEDM_ISC_LRU1000_EE_SELmask_EPC_CL1_GEN2_MASK_MODE_TRUNC	
FEDM_ISC_LRU1000_EE_SELmask_EPC_CL1_GEN2_MASK_MODE_BANK	
FEDM_ISC_LRU1000_EE_SELmask_EPC_CL1_GEN2_MASK_START_PTR	
FEDM_ISC_LRU1000_EE_SELmask_EPC_CL1_GEN2_MASK_MSB	
FEDM_ISC_LRU1000_EE_SELmask_ISO18000_6_B_MASK_LGT	Constants for Selection Mask in the reader configuration for the transponder type ISO18000-6-B
FEDM_ISC_LRU1000_EE_SELmask_ISO18000_6_B_MASK_MODE	
FEDM_ISC_LRU1000_EE_SELmask_ISO18000_6_B_MASK_START_PTR	
FEDM_ISC_LRU1000_EE_SELmask_ISO18000_6_B_MASK	

V2.03.05

- Support of the new HF-Reader ID ISC.LR2000
- Extensions for the UHF-Reader ID ISC.LRU1000 concerning the configuration
- Support for the new transponder types: HF-Transponder Innovision Jewel and UHF-Transponder EPC Class0/0+
- New communication methods in **FedmlscReader**:
 1. sendProtocol (byte cmdByte, String requestData)
 2. sendTransparent (String requestProtocol, boolean calcCrc)
- New communication functions in **FedmlscReader**:
 3. sendProtocol (byte cmdByte, string RequestData, out string ResponseData)
 4. sendTransparent (string SndProtocol, bool CalcChecksum, out string RecProtocol)
- Support for the new protocol [0x72] Set Output

- New common constants:

Constant	Comment
FEDM_ISC_TMP_B0_MODE_CINF	Flag Card Information in Mode-Byte for [0xB0][0x25] Select
FEDM_ISC_TMP_B0_MODE_WR_NE	Flag Write-Erase in Mode-Byte for [0xB0][0x24] Write Multiple Blocks
FEDM_ISC_TMP_B0_RSP_FORMAT	Format Byte in response protocol of [0xB0][0x25] Select, if CINF-Flag is set
FEDM_ISC_TMP_B2_REQ_MF_CMD	Parameter for [0xB2][0x30] Mifare Value Commands
FEDM_ISC_TMP_B2_REQ_OP_VALUE	
FEDM_ISC_TMP_B2_REQ_DEST_ADR	
FEDM_ISC_TMP_ADV_BRM_TRDATA2	2. Byte of TR-DATA in response protocol of [0x22] Read Bufer
FEDM_ISC_TMP_ADV_BRM_TRDATA2_...	Flags in 2. Byte of TR-DATA in response protocol of [0x22] Read Bufer
FEDM_ISC_TMP_0x72_OUT...	Constants for [0x72] Set Output

- Modified common constants:

Old Constant	New Constant
FEDM_ISC_TMP_ADV_BRM_TRDATA1	FEDM_ISC_TMP_ADV_BRM_TRDATA
FEDM_ISC_TMP_ADV_BRM_TRDATA1_...	FEDM_ISC_TMP_ADV_BRM_TRDATA_...

V2.03.02

- Extensions for the Reader ID ISC.MR200
- Error correction in FedmIscReaderConst. All applications linked with OBIDISC4NET must be recompiled to prevent exceptions!

V2.03.00

- Support of USB-Reader with new protocol frames
- Support of the new Reader: ID ISC.MR200
- Extensions for the Reader ID ISC.LRU1000
- Support for function units ID ISC.ANT.MUX and ID ISC.DAT
- Support of new UHF transponder types
- New functions for exchanging protocols: SendProtocol and SendTransparent
- Integration of new protocols
- Modifications in function parameters:

class	old function	new function
FedmBrmTableItem	GetData(string, out int)	GetData(string, out uint)
FedmBrmTableItem	GetData(long, out int)	GetData(long, out uint)

class	old function	new function
FedmBrmTableItem	SetData(string, int)	SetData(string, uint)
FedmBrmTableItem	SetData(long, int)	SetData(long, uint)
FedmIsoTableItem	GetData(string, out int)	GetData(string, out uint)
FedmIsoTableItem	GetData(long, out int)	GetData(long, out uint)
FedmIsoTableItem	SetData(string, int)	SetData(string, uint)
FedmIsoTableItem	SetData(long, int)	SetData(long, uint)

V1.00.00

- first release