

# Java Class Library

# ID ISC.SDK.Java

Version 4.09.00

**Software-Support for**

**OBID i-scan®**

**and**

**OBID® classic-pro**

**Reader Families**

**for 32- or 64-Bit Operating Systems**

**Windows Vista / 7 / 8 / 10**

**Linux and Android**

**with 32- or 64-Bit Java Runtime Environment (JRE) 5 or higher**

## Note

© Copyright 2003-2017 by FEIG ELECTRONIC GmbH  
Lange Straße 4  
D-35781 Weilburg-Waldhausen  
Germany  
[obid-support@feig.de](mailto:obid-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 relates 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.

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

Oracle and Java® are registered trademarks of Oracle Corporation.

Linux® is a registered Trademark of Linus Torvalds.

Android is a trademark of Google Inc.

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

Tag-it (TM) is a registered Trademark of Texas Instruments Inc.

Electronic Product Code™ (EPC) 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 an agreement between you and FEIG ELECTRONIC GmbH (hereafter "FEIG") for use of provided software **ID ISC.SDK.Java** (application programs, program libraries, source code examples and the connected documents), hereafter called licensing material. By installing and using the licensing material you agree to all terms and conditions of this agreement without exception and without limitation. If you are not or not completely in agreement with the terms and conditions, you may not install the licensing material or use it in any way.

### §1 Subject and Scope of the Agreement

1. FEIG grants you the non-exclusive and non-transferable right to install the licensed material 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 Java library OBIDISC4J.JAR, as well as the necessary native libraries, for developing your own application program and to sell this Java library OBIDISC4J.JAR, 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 Java library OBIDISC4J.JAR, 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. The licensed material is also protected by German copyright law, International Treaty provisions and the laws of the country in which it is used. Structure, organization and code of the licensed material are the valuable trade secrets of FEIG and its suppliers.
2. You agree not to change, 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 protection marks, such as copyright marks and other legal restrictions in the licensing material, you agree to keep these unchanged and to use them unchanged in all complete or partial copies which you make.
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 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 the use of the licensed material may result 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. FEIG provides the software "as is" and without any warranty. FEIG cannot guarantee the performance or the results you obtain from using the licensed material. FEIG assumes no liability or guarantee that the protection rights of third parties are not violated, nor that the software is suitable for a 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 final clause

1. This Agreement contains the complete licensing terms and conditions and supercedes any prior agreements and terms. Changes and additions must be made in writing.
2. If any provision in this agreement is declared to be void, or if for any reason is declared to be invalid or of no effect, the remaining provisions shall be in no manner affected thereby but shall remain in full force and effect. Both parties agree to replace the invalid provision with one which comes closest to its original intention.
3. This agreement is subject to the laws of the Federal Republic of Germany. Place of jurisdiction is Frankfurt a. M.

Please direct any questions pertaining to this agreement to:

FEIG ELECTRONIC GmbH  
Lange Strasse 4,  
D-35781 Weilburg-Waldhausen  
- Germany -  
Fon: +49 6471 / 3109-0  
Fax: +49 6471 / 3109-99  
e-mail: [info@feig.de](mailto:info@feig.de)  
<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](mailto: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](mailto: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](mailto:eay@cryptsoft.com)). This product includes software written by Tim Hudson ([tjh@cryptsoft.com](mailto:tjh@cryptsoft.com)).

Original SSLeay License  
-----

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

This package is an SSL implementation written by Eric Young ([eay@cryptsoft.com](mailto: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 aheared 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](mailto: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](mailto:eay@cryptsoft.com))"

The word 'cryptographic' can be left out if the rouines 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](mailto: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:**

---

<b>Licensing Agreement for use of the software .....</b>	<b>3</b>
<b>Third-party Licensing agreements .....</b>	<b>5</b>
Licensing agreement of openssl organization .....	5
<b>Contents: .....</b>	<b>7</b>
<b>1. Introduction .....</b>	<b>11</b>
1.1. Overview of all software modules .....	12
1.2. Supported operating systems .....	13
1.3. System requirements .....	13
<b>2. Revisions since the previous version .....</b>	<b>14</b>
<b>3. Installation .....</b>	<b>15</b>
3.1. Installation on Development Computer .....	15
3.1.1. Windows Vista / 7 / 8 / 10 .....	15
3.1.2. Linux .....	15
3.1.3. Android .....	16
3.2. Installation on Target Computer .....	17
3.2.1. Windows Vista / 7 / 8 / 10 .....	17
3.2.2. Linux .....	17
3.2.3. Android .....	19
<b>4. Overview of the classes .....</b>	<b>20</b>
4.1. Reader class FedmlscReader .....	20
4.2. Table classes FedmlsoTableItem and FedmBrmTableItem .....	21
4.3. Class FedmlscFunctionUnit .....	23
4.4. Class FedmlscPeopleCounter .....	24
4.5. TagHandler classes .....	24
4.6. Help classes and interfaces .....	25
4.6.1. FedmlscReaderInfo .....	25
4.6.2. FedmlscReaderTime .....	25
4.6.3. FedmCprAdu .....	25
4.6.4. FedmCprCommandQueue .....	25
4.6.5. FeUsb and FeUsbScanSearch .....	25
4.6.6. FeHexConvert .....	26

4.6.7. FeMethods .....	26
4.6.8. The FelscListener interface .....	26
4.6.9. The FeTaskListener interface .....	26
4.6.10. The FedmTaskOption property .....	27
4.6.11. The Fedm interface .....	27
4.6.12. The FedmlscReaderConst interface .....	27
4.6.13. The FedmlscReaderID interface .....	27
4.6.14. Das Interface FedmlscFunctionUnitID .....	27
<b>4.7. Exception classes .....</b>	<b>28</b>
4.7.1. FedmException .....	28
4.7.2. FePortDriverException .....	28
4.7.3. FeReaderDriverException .....	28
<b>5. Basic properties of the reader class .....</b>	<b>29</b>
<b>5.1. Initializing and finalizing .....</b>	<b>29</b>
5.1.1. Initializing .....	29
5.1.2. Finalizing .....	30
<b>5.2. Administering the communications channels .....</b>	<b>30</b>
<b>5.3. Communication with the reader .....</b>	<b>31</b>
5.3.1. Synchronous communication .....	31
5.3.2. Asynchronous communication .....	33
5.3.3. Secured data transmission with encryption .....	35
5.3.3.1. Overview .....	35
5.3.3.2. Feedback of error cases .....	35
5.3.3.3. Notes for Programmers .....	36
<b>5.4. Communication with a People Counter .....</b>	<b>37</b>
<b>5.5. Data containers .....</b>	<b>38</b>
5.5.1. Data exchange .....	38
5.5.1.1. Constant data .....	38
5.5.1.2. Data type boolean .....	38
5.5.1.3. Data type byte .....	39
5.5.1.4. Data type byte[] .....	39
5.5.1.5. Data type int .....	39
5.5.1.6. Data type long .....	39
5.5.1.7. Data type String .....	39
5.5.2. Access constants for temporary protocol data .....	40
5.5.3. Reader Configuration Parameters in the Package de.feig.ReaderConfig .....	41
5.5.4. Management of the reader configuration .....	42
5.5.5. Serializing .....	45
<b>5.6. Communication with Transponders in the Host-Mode .....</b>	<b>47</b>
<b>5.7. Examples for using the method sendProtocol .....</b>	<b>49</b>
<b>5.8. Tables .....</b>	<b>57</b>



5.8.1. Examples for using the table for ISO-Host Mode.....	58
5.8.1.1. <i>Anomaly of the addressed mode</i> .....	58
5.8.1.2. <i>Examples for using the ISO table with [0xB0] Commands</i> .....	58
5.8.1.3. <i>Examples for using the ISO table with [0xB3] Commands</i> .....	67
5.8.2. Examples for using the table for Buffered Read Mode .....	69
5.9. Example for using the method <code>sendSAMCommand</code> .....	71
5.10. Example for using the method <code>sendTclApdu</code> .....	72
5.11. Example for using the method <code>sendCommandQueue</code> .....	73
5.12. Example for communicating with a People Counter.....	75
5.13. Example for use of TagHandler classes .....	76
6. Basic properties of the class <code>FedmlscFunctionUnit</code> .....	78
6.1. Initializing und Finalizing .....	78
6.1.1. Initializing .....	78
6.1.2. Finalizing.....	78
6.2. Communication with a function unit.....	79
6.3. Examples for using the method <code>sendProtocol</code> .....	80
7. Error handling .....	82
7.1. Return value .....	82
7.2. Exceptions .....	82
8. Appendix .....	83
8.1. List of error codes.....	83
8.2. Supported OBID® Readers.....	86
8.3. Supported Transponders.....	87
8.4. Revision history .....	88

## Remarks concerning the documentation for this library

This manual describes a software library for which there is also online documentation. For this reason we have intentionally avoided documenting more than absolutely necessary for understanding the functionality and utilization of the classes. It is presumed that the user of this library will refer to the online documentation for details on the classes and methods.

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 Java class library ID OBIDISC4J from FEIG ELECTRONIC GmbH represents yet another component for simplifying the development of application programs in Java for OBID *i-scan*® and OBID®*classic-pro* readers.

This manual is intended as an introduction to the library and supplements the online documentation.

The Java class library ID OBIDISC4J currently supports Windows, Linux and Android.

The Java class library ID OBIDISC4J is based on the C++ class library ID FEDM as well as the native function libraries ID FECOM, ID FEUSB, ID FETCP, ID FEISC, ID FETCL and ID FEFU. The Java class library therefore consists only of a wrapper. Nevertheless, the full functionality of the C++ class library is accessible for Java:

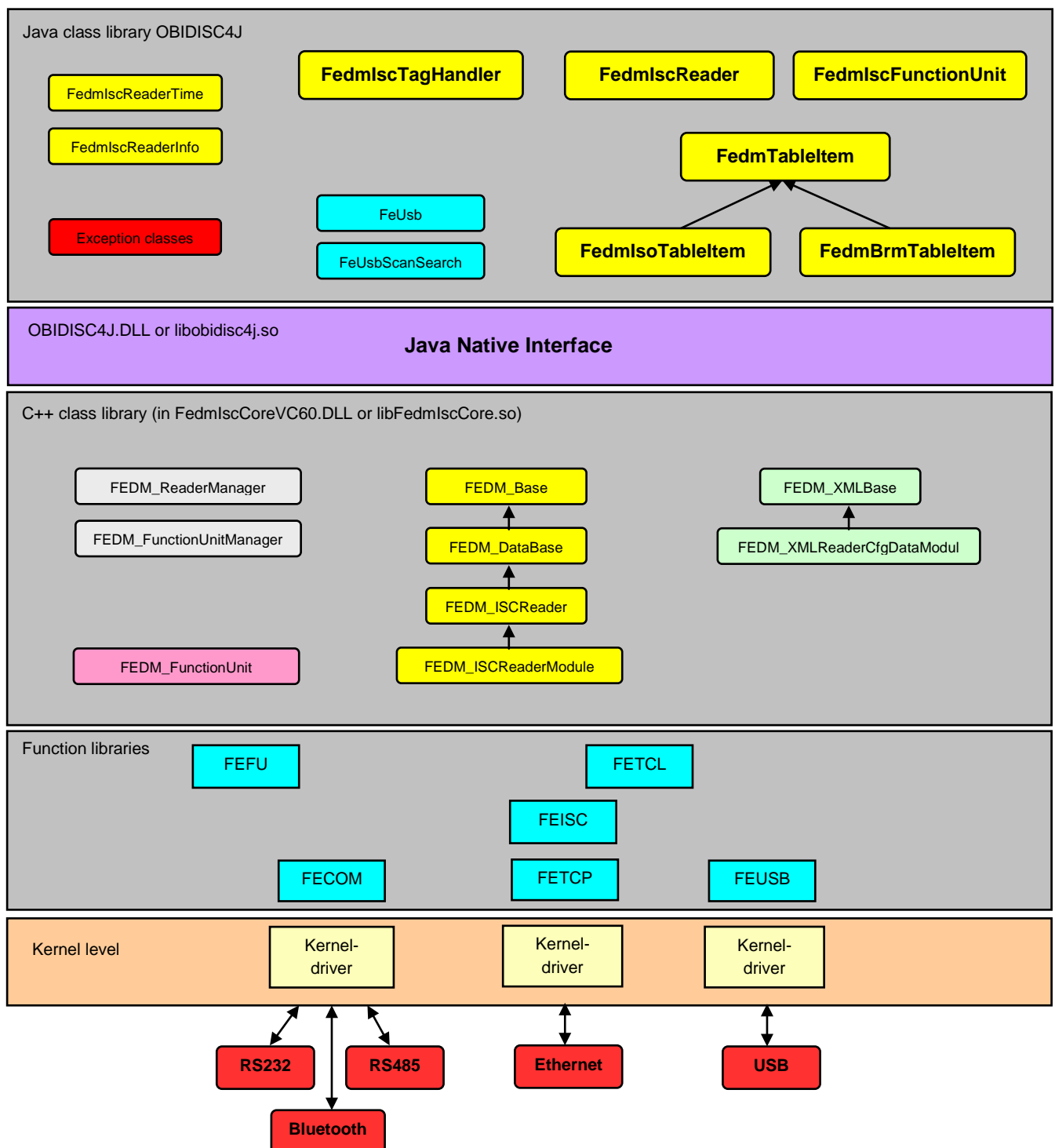
- A uniform organizational principle for saveable data from reader and transponder in data containers and tables.
- Overloaded methods for access to the data containers and tables.
- A single, easy to use communications method.
- Synchronous and asynchronous communication
- Complete error handling using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

### **Important note:**

The ID OBIDISC4J 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 OBIDISC4J Java 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 Java methods.



---

## 1.2. Supported operating systems

---

- 32- or 64-Bit Microsoft Windows Vista / 7 / 8 / 10
- Microsoft Windows CE 6 upon request
- 32- or 64-Bit Linux (x86 processor only, others upon request)
- Android V4.0.3 or higher with ARMv5, ARMv7, x86 or MIPS Architecture

---

## 1.3. System requirements

---

- 32 or 64-Bit Java Runtime Environment (JRE) 5 or higher
- Android: Version 4.0.3 or higher. Android-SDK in revision 15 or higher. USB support requires a rooted Android system.

## 2. Revisions since the previous version

---

- Update of namespaces and access constants for reader configuration
- 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 **FedmlscTagHandler\_ISO14443\_4\_MIFARE\_DESFire** for getValue
- 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

---

---

### 3.1. Installation on Development Computer

---

---

#### 3.1.1. Windows Vista / 7 / 8 / 10

---

The following settings for the Netbeans sample projects must be done:

- The "Working Directory" (Project Properties → Run) must point to the path of the native libraries (sw-run\windows\...).
- Use the proper "Java Platform" (Project Properties → Libraries) for the native libraries (i.e. x86 or x64).

If an alternative IDE is preferred, the installation is the same as for a target computer and described in [3.2.1. Windows Vista / 7 / 8 / 10](#).

---

#### 3.1.2. Linux

---

The installation is the same as for a target computer and described in [3.2.2. Linux](#).

### 3.1.3. Android

The App development in Java for Android requires an IDE like Eclipse with installed Android-SDK (API revision 15 or higher).

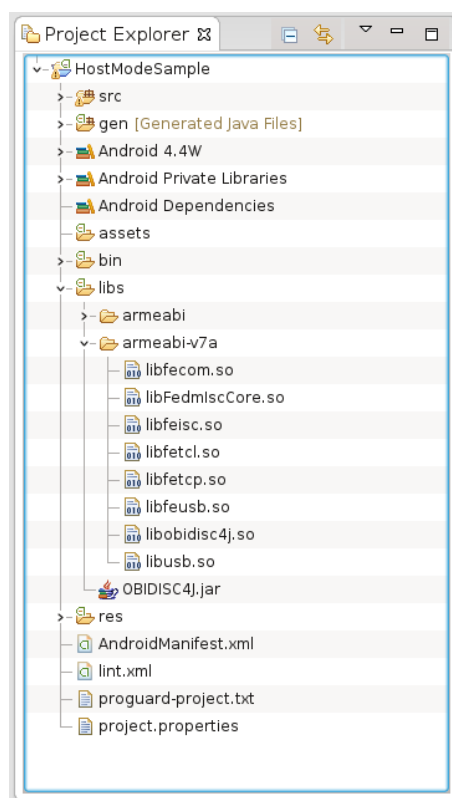
NOTE: USB support requires a rooted Android system.

The following files are included in this SDK:

File	Description
libfecom.so	Native library for serial interface
libfetcp.so	Native library for TCP/IP interface
libusb.so	Native library for USB interface
libfeusb.so	Native library for USB interface
libfeisc.so	Native library for OBID i-scan® and OBID® classic-pro Reader
libfetcl.so	Native library for ISO14443-4 T=CL protocols with OBID® classic-pro Reader
libFedmiscCore.so	Native library for OBID i-scan® and OBID® classic-pro Reader
libobidisc4j.so	Native library for OBID i-scan® and OBID® classic-pro Reader
OBIDISC4J.jar	Java library

In your App project you have to create a subdirectory `libs/armeabi`. Copy into this directory all `.so`-files. Copy also the Java library file `OBIDISC4J.jar` into the directory `libs`.

An example for a project is shown with the following picture:





## 3.2. Installation on Target Computer

### 3.2.1. Windows Vista / 7 / 8 / 10

The following library files are located under `sw-run\windows\...`:

File	Description
FECOM.DLL	Native library for serial interface
FETCP.DLL	Native library for TCP/IP
FEUSB.DLL	Native library for USB
FEISC.DLL	Native library for OBID i-scan® and OBID® classic-pro Reader
FEFU.DLL	Native library for OBID i-scan® external Function Units
FETCL.DLL	Native library for ISO14443-4 T=CL protocols with OBID® classic-pro Reader
OBIDISC4J.DLL	Native library for OBID i-scan® and OBID® classic-pro Reader
OBIDISC4J.jar	Java library

Installation is quite simple:

Copy all DLL files to the directory: *Java-directory\jre\bin*

Copy the file OBIDISC4J.jar to the directory: *Java-directory\jre\lib\ext*

Alternately you can select any other directory, as long as you tell this to the Java environment: `java -classpath Directory`.

### 3.2.2. Linux

The following library files are located under `sw-run\linux\...`:

File <sup>1</sup>	Description
libfecom.so.x.y.z	Native library for serial interface
libfeusb.so.x.y.z	Native library for USB
libfetcp.so.x.y.z	Native library for TCP/IP
libfeisc.so.x.y.z	Native library for OBID i-scan® and OBID® classic-pro Reader
libfefu.so.x.y.z	Native library for OBID i-scan® external Function Units
libfetcl.so.x.y.z	Native library for ISO14443-4 T=CL protocols with OBID® classic-pro Reader
libFedmlscCore.so.x.y.z	Native library for OBID i-scan® and OBID® classic-pro Reader
libobidisc4j.so.x.y.z	Native library for OBID i-scan® and OBID® classic-pro Reader
OBIDISC4J.jar	Java library

<sup>1</sup> x.y.z represents the version number of the library file

Installation is quite simple:

Change to the SDK library path (`sw-run\linux\...`) and call the script `install-libs.sh` with the destination install path for the libraries as parameter (e.g. `/usr/lib`).

This will copy all library file to the destination path and generate the necessary links:

```
ln -sf lib*.so.x.y.z lib*.so.x
```

```
ln -sf lib*.so.x lib*.so
```

Finally `ldconfig` is called.

Copy the `OBIDISC4J.jar` file to the directory: *Java-directory\jre\lib\ext*

Alternately you can select any other directory, as long as you tell this to the Java environment: `java -classpath Directory`.

In the case that the JVM cannot find the native library `libobidisc4j.so`, set a symbolic link to the library in the same directory where `OBIDISC4J.jar` is located.

If USB is intended to be used, then additional installation steps are necessary. Please follow the installation instructions in the FEUSB manual (`H00501-#e-ID-B.pdf`).

### 3.2.3. Android

---

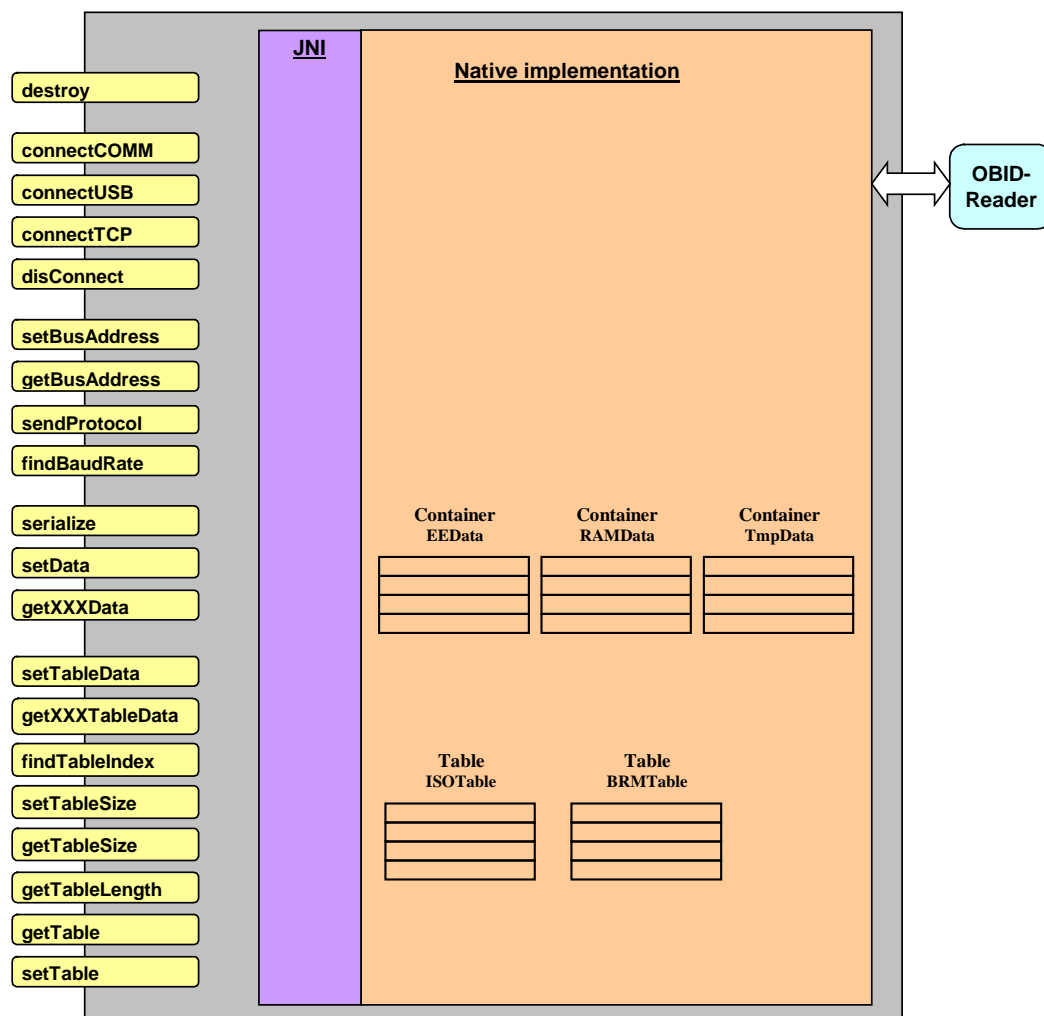
Create an Installation file (.apk file) within Eclipse and execute this file on the target.

## 4. Overview of the classes

### 4.1. Reader class FedmlscReader

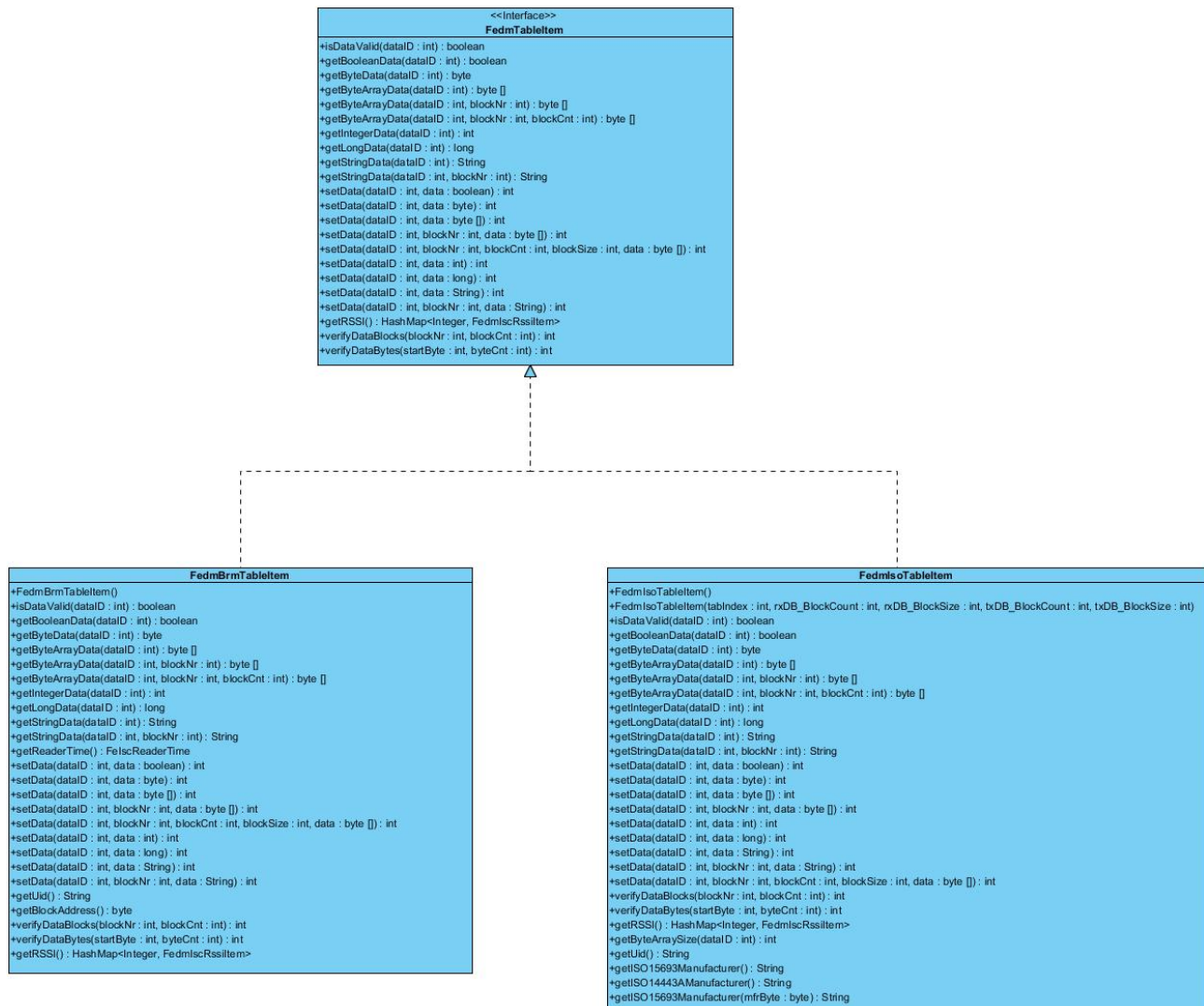
The reader class **FedmlscReader** is the main class of the Java library. The component diagram shows an overview of the reader class.

Only the most important methods are shown. Attributes are not contained in the Java class. Refer to the online documentation 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 *getXXXTableData*<sup>2</sup> and *setTableData* of the reader class **FedmIscReader**. 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.

<sup>2</sup> XXX stands for Boolean, Byte, Integer, Long, String and represents the data types of the return value.

**FedmBrmTableItem** contains transponder data that were read by the reader in Buffered Read Mode or Notification Mode. Since both modes are purely read modes, no data can be written in **FedmBrmTableItem** using *setData*.

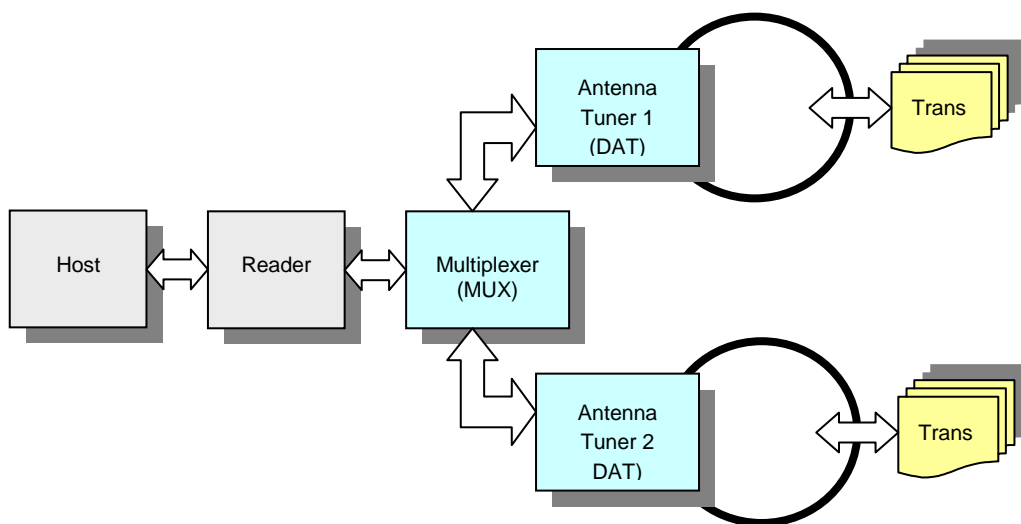
The table classes are always container for read data or data prepared to be written. The data exchange with the Transponder is always executed with the method *sendProtocol* of the reader class **FedmlscReader**.

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

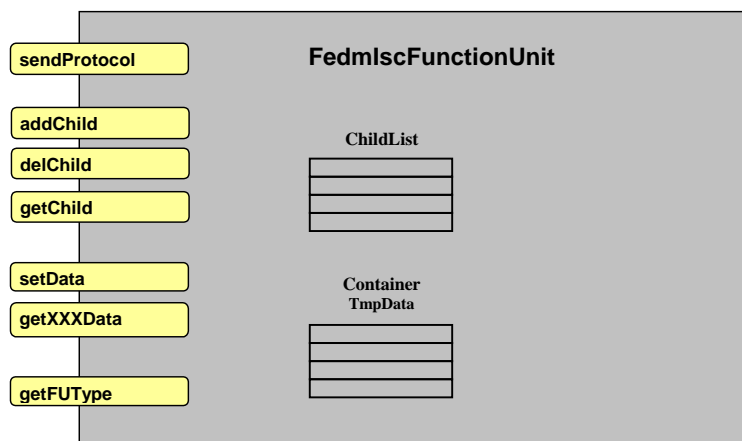
The class **FedmlscFunctionUnit** represents an external function unit (FU) integrated in the antenna cable of the reader. The class has no base class. For a deeper understanding of the possibilities of function units you should read the system manual H30701-xe-ID-B (HF) or H80302-xe-ID-B (UHF). Additional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

The component diagram shows an overview of the function unit class.



---

#### 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 [5.4. Communication with a People Counter](#).

---

#### 4.5. TagHandler classes

---

TagHandler classes are specific to Transponder standards likeISO 14443, ISO 15693 and EPC Class 1 Gen 2 or customizedformanufacturer specific extendedAPI. Each standard and chip type is implemented as a class and all classes together build a hierarchical system ofderived classes. Base class is FedmlscTagHandler.

More information can be found in [5.6.Communication with Transponders in the Host-Mode](#)



---

## 4.6. Help classes and interfaces

---

---

### 4.6.1. FedmlscReaderInfo

---

**FedmlscReaderInfo** is a class collecting 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.



---

### 4.6.2. FelscReaderTime

---

**FelscReaderTime** 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**.

---

### 4.6.3. FedmCprApdu

---

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

---

### 4.6.4. FedmCprCommandQueue

---

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

---

### 4.6.5. 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.

---

#### 4.6.6. FeHexConvert

---

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

---

#### 4.6.7. FeMethods

---

The class **FeMethods** contains useful methods for extracting data values from the access constants (s. [5.5.2.Access constants for temporary protocol data](#)).

---

#### 4.6.8. 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.

---

#### 4.6.9. The FeTaskListener interface

---

The **FeTaskListener** interface enables event handling from the native library. With this interface, the transponder or reader data of an asynchronous read operation can be queried.

---

#### 4.6.10. The FedmTaskOption property

---

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

---

#### 4.6.11. The Fedm interface

---

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

---

#### 4.6.12. The FedmlscReaderConst interface

---

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

---

#### 4.6.13. The FedmlscReaderID interface

---

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

---

#### 4.6.14. Das Interface FedmlscFunctionUnitID

---

The interface **FedmlscFunctionUnitID** gathers all access constants for the external OBID *i-scan*® Function Units.

---

## 4.7. Exception classes

---

---

### 4.7.1. FedmException

---

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

---

### 4.7.2. FePortDriverException

---

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

---

### 4.7.3. 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 methods can be roughly divided into five categories:

- a) Methods for initializing and finalizing
- b) Methods for the communications channels
- c) Methods for 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 <i>setBusaddress</i> .  |
| 2. Table size  | The tables <i>ISOTable</i> and <i>BRMTable</i> contained in the reader class <b>FedmlscReader</b> do not have a preset size. Therefore you <b>must</b> (!) use the method <i>setTableSize</i> to dimension the required table before first communicating 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 cannot 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 *FedmiscReaderConst*.

---

### 5.1.2. Finalizing

---

In Java the garbage collector assumes the task of removing no longer needed objects. This works wonderfully in pure Java applications. But objects that were created in native code are not subject to the scrutiny of the garbage collector. Therefore the programmer must take over this work. In the class of this class library, this work is taken care of in one line: you invoke the reader class method *destroy* when you no longer need the reader object. If you omit this finalizing, you will get an exception no later than when the application is closed.

---

## 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 *FedmiscReader*: *connectCOM*, *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 communications parameters (baud rate, frame).

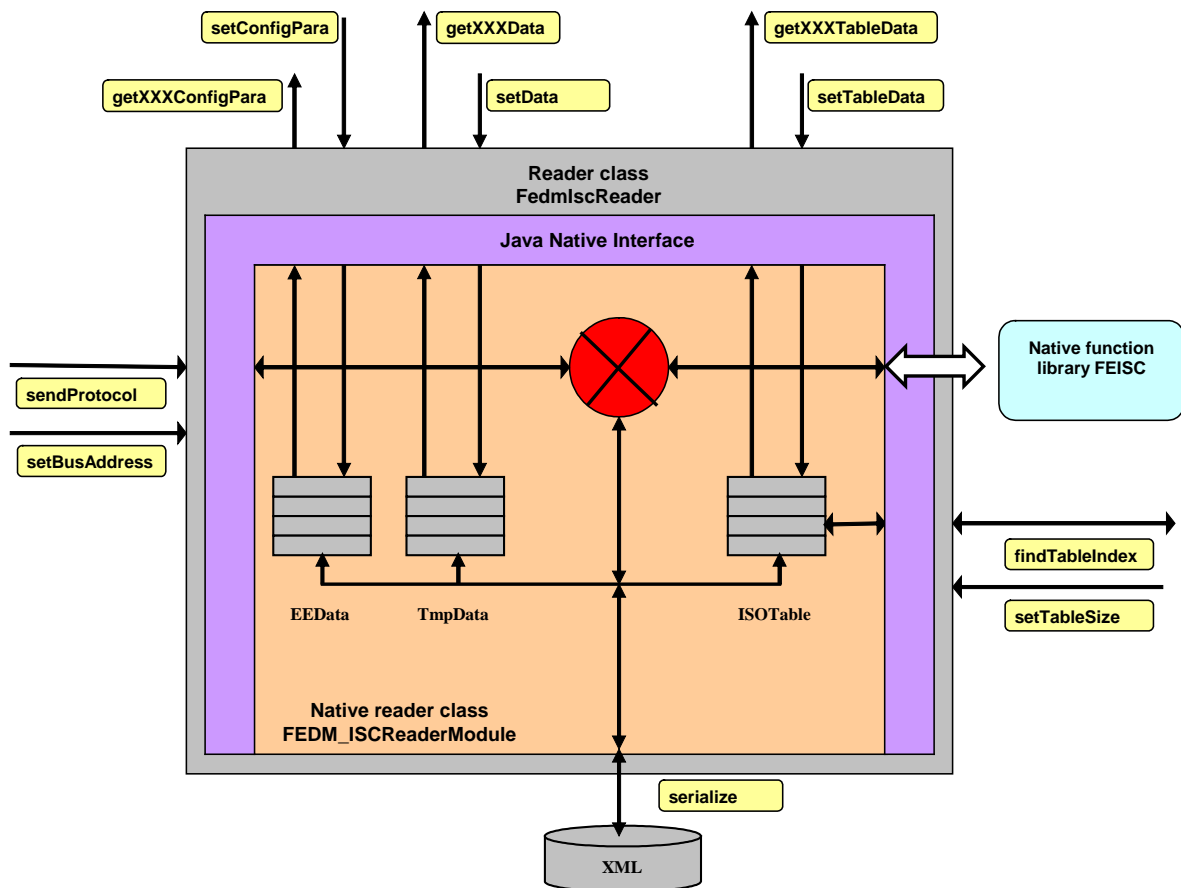
In the exceptional case that multiple USB readers have to be supported at the same time in an application, there is the class *FeUsb*, which provides special methods for this case.

**Bluetooth Readers under Linux** are administrated as */dev/rfcomm<x>*. This makes the necessity to have a special connect method called *connectBT*. With Windows the call of *connectBT* will be redirected to *connectCOMM*.

### 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) methods *getXXXData*<sup>3</sup> and *setData*, as well as *getXXXTableData* and *setTableData*. In addition, the method *serialize* is used 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 communication 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.

<sup>3</sup> XXX stands for Boolean, Byte, Integer, Long, String and represents the data types of the return value.

The key to the protocol data are so-called access constants for temporary protocol data (e.g. `FEDM_ISC_TMP_READER_INFO_MODE`) and the namespace `OBID.ReaderConfig` for reader configuration parameters (e.g. `de.feig.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 [5.5.2. Access constants for temporary protocol data](#) and [5.5.3. Reader Configuration Parameters in the Package `de.feig.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 [5.7. Examples for using the method `sendProtocol`](#). The definition of each reader configuration parameter in the package `de.feig.ReaderConfig` is documented in the system manual of the reader.

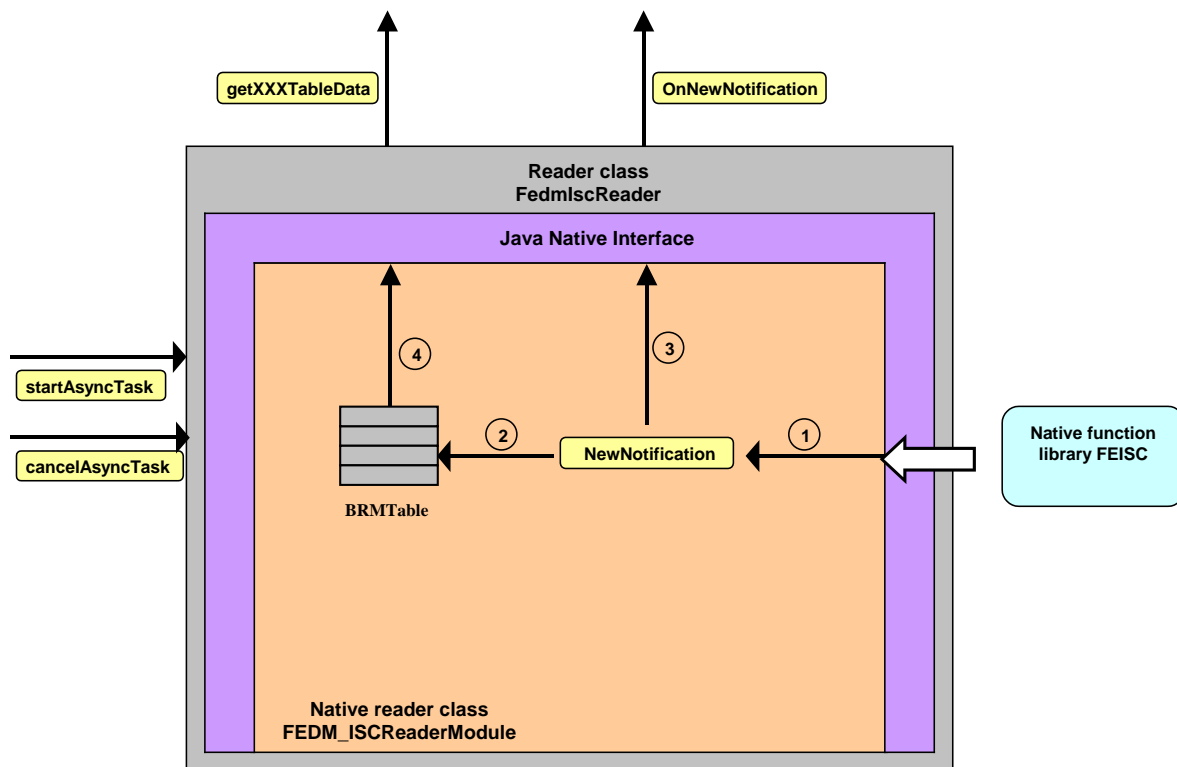
The OBID®-Readers on the serial port are bus-compatible and require the bus address in the protocol. This should be set using the method `setBusAddress`.



### 5.3.2. Asynchronous communication

The asynchronous communication is initiated amongst others by the method **startAsyncTask** of the reader class **FedmlscReader** and is triggered by notification events of the reader. An asynchronous task initialised by **startAsyncTask** can only be used if the reader supports the Notification Mode or the asynchronous option for the Inventory command in the Host Mode<sup>4</sup>. Asynchronous tasks are also launched by the methods **sendTclApdu**, **sendSAMCommand** and **sendCommandQueue**. 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 (3<sup>rd</sup> step). Inside the event method (4<sup>th</sup> 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**.

<sup>4</sup> 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

### 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 over Ethernet (TCP/IP) 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**

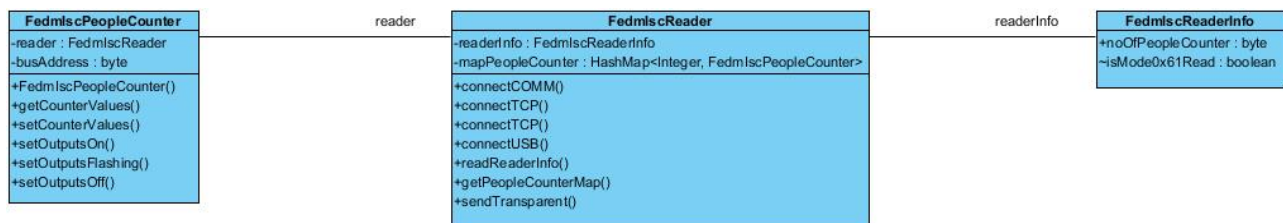
---

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.connectTCP` with authentication password is required.
4. If the Host application receives after a plain or enciphered data transmission the status 0x19 an authentication into the Reader is required.
5. If the error code -4093 or -4094 occurs in the Host application an 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 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 `HashMap`, can be retrieved with the method `getPeopleCounterMap()`. The key of the `HashMap` 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 `HashMap` 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 in the JavaDoc class reference as well as with an example in [5.12. Example for communicating with a People Counter](#).

## 5.5. 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-byte 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

### 5.5.1. Data exchange

Access to the data is possible primarily using the overloaded methods *setData* and *getXXXData*<sup>5</sup>. Each method invocation can read or write exactly one parameter, which is identified by an access constant (see [5.5.2. Access constants for temporary protocol data](#)).

Access to the configuration parameter is possible primarily using the overloaded methods *setConfigPara* and *getConfigParaAsXXX*.

The following section shows the use of *getXXXdata* and *setData* for various data types. The use of *getConfigParaAsXXX* and *setConfigPara* is analogous, but with the difference that instead of an access constant a string with the parameter name from the package `de.feig.ReaderConfig` must be passed.

#### 5.5.1.1. Constant data

```
int iErr = setData(FedmIscReaderID.FEDM_ISC_TMP_READ_CFG_MODE, false);           // boolean
int iErr = setData(FedmIscReaderID.FEDM_ISC_TMP_READER_INFO_MODE, (byte)1);      // byte
int iErr = setData(FedmIscReaderID.FEDM_ISC_TMP_READER_INFO_MODE, (long)134);    // long
int iErr = setData(FedmIscReaderID.FEDM_ISC_TMP_READER_INFO_MODE, "0134");      // String
```

#### 5.5.1.2. Data type boolean

```
boolean data = false;
data = getBooleanData(FedmIscReaderID.FEDM_FEDM_ISC_TMP_INP_STATE_IN1);
int iErr = setData(FedmIscReaderID.FEDM_ISC_TMP_READ_CFG_MODE, data);
```

<sup>5</sup> XXX stands for Boolean, Byte, Integer, Long, String and represents the data types of the return value.

---

**5.5.1.3. Data type byte**

---

```
byte data = 1;
data = getByteData(FedmIsedReaderID.FEDM_ISC_TMP_INP_STATE);
int iErr = setData(FedmIsedReaderID.FEDM_ISC_TMP_READER_INFO_MODE, data);
```

---

---

**5.5.1.4. Data type byte[]**

---

```
byte[] data;
data = getByteArrayData(FedmIsedReaderID.FEDM_ISC_TMP_SOFTVER_SW_REV);
iErr = setData(FedmIsedReaderID.FEDM_ISCLR_TMP_READER_PW, data);
```

---

---

**5.5.1.5. Data type int**

---

```
int data = 0;
data = getData(FedmIsedReaderID.FEDM_ISC_TMP_B0_RSP_DB_EXT_ADR_E);
int iErr = setData(FedmIsedReaderID.FEDM_ISC_TMP_B0_REQ_DB_ADR_EXT, data);
```

---

---

**5.5.1.6. Data type long**

---

```
long data = 0;
data = getLongData(FedmIsedReaderID.FEDM_ISC_TMP_INP_STATE);
int iErr = setData(FedmIsedReaderID.FEDM_ISC_TMP_READER_INFO_MODE, data);
```

---

---

**5.5.1.7. Data type String**

---

ALL data that are read using a *getStringData* 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** (see [4.6.6. FeHexConvert](#)) is provided for converting string values into other data types or the reverse.

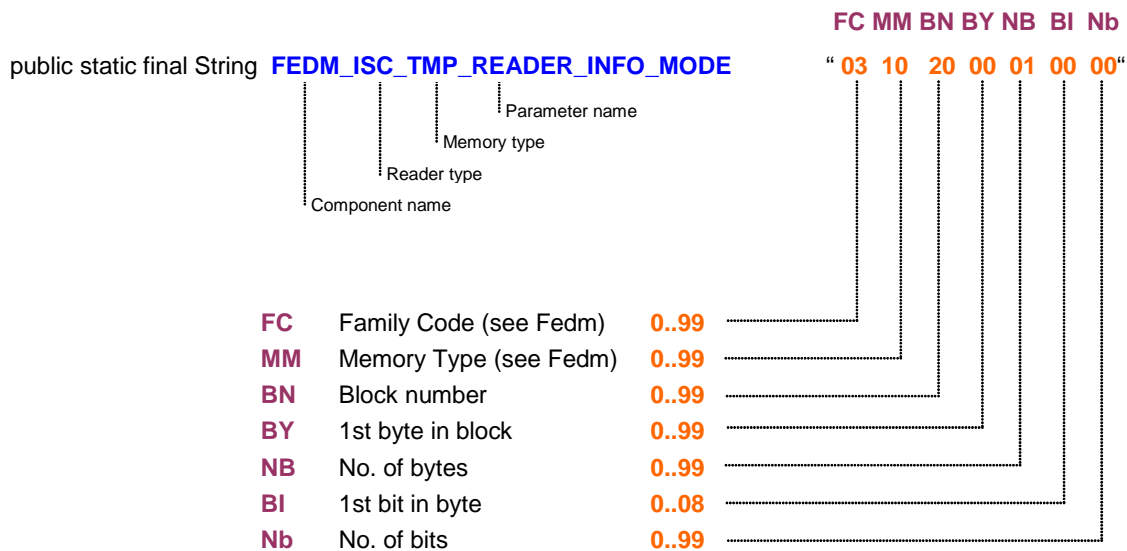
To convert numerical values into string, which in the above example make up the number „159“, the Java library methods are recommended.

```
String data;
data = getStringData(FedmIsedReaderID.FEDM_ISC_TMP_INP_STATE);
int iErr = setData(FedmIsedReaderID.FEDM_ISC_TMP_READER_INFO_MODE, data);
```

### 5.5.2. 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:



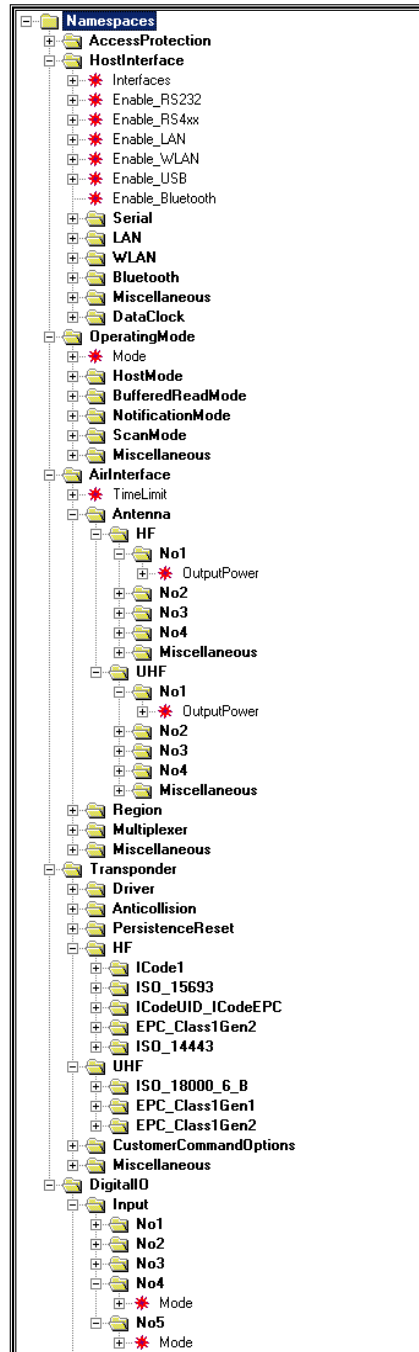
These access constants are used exclusively with the methods *setData* and *getXXXData*. The access constant says nothing about the data type of a parameter. This is determined only by the data type of the access method. One can therefore read the bus address in the above example either as an integer or as a string or some other plausible data type (see [5.5.1. Data exchange](#)).

All access constants are contained in the interface **FedmlscReaderID**.



### 5.5.3. Reader Configuration Parameters in the Package de.feig.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 package de.feig.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 package de.feig.ReaderConfig

#### 5.5.4. 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
int IdleModeRel1          // Parameter IDLE-MODE

// Defaults for the next sendProtocol
reader.setData(FEDM_ISC_TMP_READ_CFG, (byte)0x00); // reset everything
reader.setData(FEDM_ISC_TMP_READ_CFG_ADR, CfgAdr); // set address
reader.setData(FEDM_ISC_TMP_READ_CFG_LOC, EEPROM); // set memory location on RAM

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

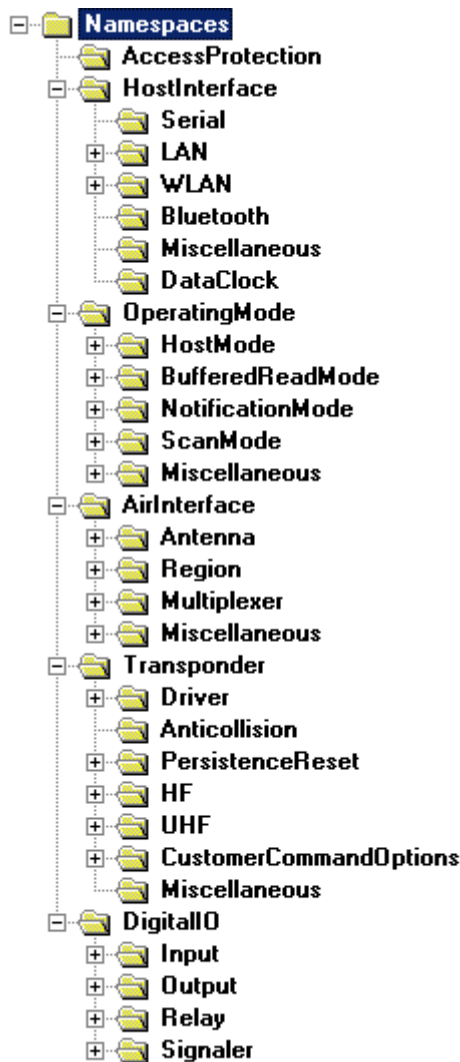
IdleModeRel1 = 3;          // REL1 alternating on (Note: set frequency in Parameter IDLE-FLASH)

reader.setConfigPara(de.feig.ReaderConfig.DigitalIO.Relay.No1.IdleMode, IdleMode, false); // change value in RAM
```

```
// Defaults for the next sendProtocol
reader.setData(FEDM_ISC_TMP_WRITE_CFG, (byte)0x00); // reset everything
reader.setData(FEDM_ISC_TMP_WRITE_CFG_ADR, CfgAdr); // set address
reader.setData(FEDM_ISC_TMP_WRITE_CFG_LOC, EEPROM); // set memory location on EEPROM

// rewrite configuration data
reader.sendProtocol(0x81);
```

The methods *getConfigParaAsXXX* and *setConfigPara* receive a string with parameter name from the package *de.feig.ReaderConfig*. This package contains further interfaces 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 interface names.



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.

### 5.5.5. Serializing

The integrated serializing method *serialize* allows saving of 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="00" 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"/>
<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")
```

## 5.6.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. [5.8.1. Examples for using the table for ISO-Host Mode](#))
2. TagHandler API, based on specialized Transponder classes (s. [5.13. Example for use of TagHandler classes](#))

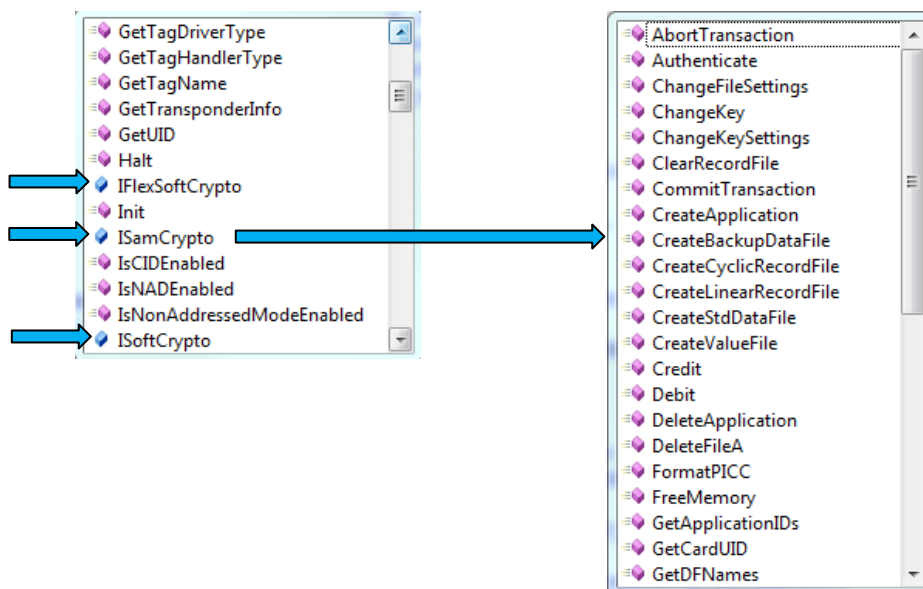
It is recommended to use the 2nd API for new projects. TagHandler classes are specific to Transponder standards like ISO 14443, ISO 15693 and EPC Class 1 Gen 2 or customized for manufacturer specific extended API. Each standard and chip type is implemented as a class and all classes together build a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

Precondition for the use of TagHandler classes is 1<sup>st</sup> the use of the methods `TagInventory` and `TagSelect` from the Reader class `FedmlscReader` and 2<sup>nd</sup> 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.



H31101-e-ID-B.docx





## 5.7. Examples for using the method `sendProtocol`

The method `sendProtocol` of the reader class is vitally important for the protocol transfer. For this reason an example is shown for each control byte, 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 interface **FedmlscReaderID** 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.

In the examples below it is assumed that the reader class **FedmlscReader** and the interfaces **FedmlscReaderID** and **FedmlscReaderConst** are incorporated:

```
import de.feig.*;
```

The reader object shall be defined as:

```
FedmIscReader reader = new FedmIscReader;
```

[Control Byte] Protocol	Example
[0x18] Destroy EPC	<pre>byte mode = 0;           // mode (always 0) byte len = 0;            // number of bytes in EPC byte[] pw = new byte[3]; // password byte[] epc = new byte[32]; // buffer for EPC  // take the data e.g. from input fields  reader.setData(FEDM_ISC_TMP_EPC_DESTROY_MODE, (byte)0); reader.setData(FEDM_ISC_TMP_EPC_DESTROY_LEN, len); reader.setData(FEDM_ISC_TMP_EPC_DESTROY_PASSWORD, pw, 3); reader.setData( FeMethod.getAdrOfId( FEDM_ISC_TMP_DESTROY_EPC, 16), epc, len,                TMP_DATA_MEM);  reader.sendProtocol((byte)0x18);</pre>
[0x1A] Halt	<pre>reader.sendProtocol((byte)0x1A);</pre>
[0x1B] Reset QUIET Bit	<pre>reader.sendProtocol((byte)0x1B);</pre>
[0x1C] EAS	<pre>reader.sendProtocol((byte)0x1C);</pre>

[Control Byte] Protocol	Example
[0x21]Read Buffer	<pre> byte dataSets = 1;    // Number requested Data sets byte trData = 0;      // Data set structure byte recSets = 0;     // Number of Data sets in protocol  reader.setData(FEDM_ISCLR_TMP_BRM_SETS, dataSets);  reader.sendProtocol((byte)0x21); // read data from transponder using Buffered Read Mode  trData =reader.getBytesData(FEDM_ISCLR_TMP_BRM_TRDATA); recSets =reader.getBytesData(FEDM_ISCLR_BRM_RECSETS);  // All other transponder datas are enclosed in the m_BRMTable. Example for data access in // 5.8.2. Examples for using the table for Buffered Read Mode </pre>
[0x22]Read Buffer	<pre> int dataSets = 1;    // Number requested Data sets byte trData = 0;     // Data set structure int recSets = 0;     // Number of Data sets in protocol  reader.setData(FEDM_ISC_TMP_ADV_BRM_SETS, dataSets);  reader.sendProtocol((byte)0x22); // read data from transponder using Buffered Read Mode  trData =reader.getBytesData(FEDM_ISC_TMP_ADV_BRM_TRDATA1); recSets =reader.getIntegerData(FEDM_ISC_ADV_BRM_RECSETS);  // All other transponder datas are enclosed in the m_BRMTable. Example for data access in // 5.8.2. Examples for using the table for Buffered Read Mode </pre>
[0x31] Read Data Buffer Info	<pre> int tabSize = 0;    // Size of the data buffer int tabStart = 0;   // Start address for the first data set int tabLen = 0;     // Number of Data sets in the data buffer  reader.sendProtocol((byte)0x31);  tabSize =reader.getIntegerData(FEDM_ISCLR_TMP_TAB_SIZE); tabStart =reader.getIntegerData(FEDM_ISCLR_TMP_TAB_START); tabLen =reader.getIntegerData(FEDM_ISCLR_TMP_TAB_LEN); </pre>
[0x32] Clear Data Buffer	<pre> reader.sendProtocol((byte)0x32); </pre>
[0x33] Initialize Buffer	<pre> reader.sendProtocol((byte)0x33); </pre>
[0x34] Force Notify Trigger	<pre> reader.sendProtocol((byte)0x34); </pre>
[0x52] Baud Rate Detection	<pre> reader.sendProtocol((byte)0x52); </pre>
[0x55] Start Flash Loader	<pre> reader.sendProtocol((byte)0x55); </pre>
[0x63] CPU Reset	<pre> reader.sendProtocol((byte)0x63); </pre>
[0x64] System Reset	<pre> byte mode = 0;    // LRU1000 RF-Controller (1 for LRU1000 AC-Controller)  reader.setData(FEDM_ISC_TMP_SYSTEM_RESET_MODE, mode);  reader.sendProtocol((byte)0x64); </pre>
[0x65] Get Software Version	<pre> String softVer = new String; // Software-version as string  reader.sendProtocol((byte)0x65);  softVer =reader.getStringData(FEDM_ISC_TMP_SOFTVER); </pre>
[0x66] Get Reader Info	<pre> String softVer = new String; // Software-version as string  reader.SetData(FEDM_ISC_TMP_READER_INFO_MODE, (int)0); // identical with [0x65] </pre>

[Control Byte] Protocol	Example
	<pre>//reader.setData(FEDM_ISC_TMP_READER_INFO_MODE, (int)1); // LRU1000: ACC reader.sendProtocol((byte)0x66); reader.getData(FEDM_ISC_TMP_SOFTVER, softVer); // identical with [0x65] //reader.getData(FEDM_ISC_TMP_FIRMWARE_VERSION, softVer); // LRU1000: ACC</pre>
[0x69] RF Reset	<pre>reader.sendProtocol((byte)0x69);</pre>
[0x6A] RF ON/OFF	<pre>byte RF = 1; // RF ON reader.setData(FEDM_ISC_TMP_RF_ONOFF, RF); reader.sendProtocol((byte)0x6A);</pre>
[0x6C] Set Noise Level	<pre>int NLMin = 500; // minimum Noise Level int NLAvg = 1000; // average Noise Level int 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((byte)0x6C);</pre>
[0x6D] Get Noise Level	<pre>int NLMin = 0; // minimum Noise Level int NLAvg = 0; // average Noise Level int NLMax = 0; // maximum Noise Level  reader.sendProtocol((byte)0x6D);  NLMin = reader.getIntegerData(FEDM_ISC_TMP_NOISE_LEVEL_MIN); NLAvg = reader.getIntegerData(FEDM_ISC_TMP_NOISE_LEVEL_AVG); NLMax = reader.getIntegerData(FEDM_ISC_TMP_NOISE_LEVEL_MAX);</pre>
[0x6E] Reader Diagnostic	<pre>byte diagMode = 1; // Diagnostic-Mode reader.setData(FEDM_ISCLR_TMP_DIAG_MODE, diagMode); reader.sendProtocol((byte)0x6E);</pre>
[0x6F] Base Antenna Tuning	<pre>reader.sendProtocol((byte)0x6F); // the Long-Range-Reader changes into the special mode // the mode can be left only by performing a reset</pre>
[0x72] Set Output	<pre>// example from system manual ID ISC.LRU1000  reader.setData(FEDM_ISC_TMP_0x72_OUT_MODE, (byte)0x00); // set mode to 0  reader.setData(FEDM_ISC_TMP_0x72_OUT_N, (byte)0x03); // activate 3 outputs reader.setData(FEDM_ISC_TMP_0x72_OUT_NR_1, (byte)0x01); // output 1 reader.setData(FEDM_ISC_TMP_0x72_OUT_TYPE_1, (byte)0x00); // type: general output reader.setData(FEDM_ISC_TMP_0x72_OUT_MODE_1, (byte)0x03); // alternating reader.setData(FEDM_ISC_TMP_0x72_OUT_FREQ_1, (byte)0x01); // 4 Hz reader.setData(FEDM_ISC_TMP_0x72_OUT_TIME_1, (int)5); // 500 ms  reader.setData(FEDM_ISC_TMP_0x72_OUT_NR_2, (byte)0x01); // relay 1 reader.setData(FEDM_ISC_TMP_0x72_OUT_TYPE_2, (byte)0x04); // type: relay reader.setData(FEDM_ISC_TMP_0x72_OUT_MODE_2, (byte)0x02); // switching off reader.setData(FEDM_ISC_TMP_0x72_OUT_FREQ_2, (byte)0x00); // unchanged reader.setData(FEDM_ISC_TMP_0x72_OUT_TIME_2, (int)2); // 200 ms  reader.setData(FEDM_ISC_TMP_0x72_OUT_NR_3, (byte)0x02); // relay 2 reader.setData(FEDM_ISC_TMP_0x72_OUT_TYPE_3, (byte)0x04); // type: relay reader.setData(FEDM_ISC_TMP_0x72_OUT_MODE_3, (byte)0x01); // switching on</pre>

[Control Byte] Protocol	Example
	<pre> reader.setData(FEDM_ISC_TMP_0x72_OUT_FREQ_3, (byte)0x00); // unchanged reader.setData(FEDM_ISC_TMP_0x72_OUT_TIME_3, (int)10);      // 1000 ms  reader.sendProtocol((byte)0x72); </pre>
[0x71] Set Output	<pre> // Example 1 from the system manual ID ISC.M01  reader.setData(FEDM_ISCM_TMP_OUT_OS, (int)0);           // OS-Bytes reset reader.setData(FEDM_ISCM_TMP_OUT_OS_OUT1, (byte)0x01); // Output 1 active reader.setData(FEDM_ISCM_TMP_OUT_OS_LED_G, (byte)0x10); // LED green off reader.setData(FEDM_ISCM_TMP_OUT_OS_LED_R, (byte)0x01); // LED red on reader.setData(FEDM_ISCM_TMP_OUT_OS_BEEPER, (byte)0x11); // Beeper alternated on  reader.setData(FEDM_ISCM_TMP_OUT_OSF, (int)0);         // OSF-Bytes reset reader.setData(FEDM_ISCM_TMP_OUT_OSF_BEEPER, (byte)0x01); // Beeper with 4Hz  reader.setData(FEDM_ISCM_TMP_OUT_OSTIME, (int)5);      // 500ms active time Beeper and LED's reader.setData(FEDM_ISCM_TMP_OUT_OUTTIME, (int)3);    // Output 1 300ms active  reader.sendProtocol((byte)0x71); </pre>
[0x74] Get Input	<pre> // Example for ID ISC.LR  bool in1 = false; // Input 1 bool in2 = false; // Input 2 bool dip1 = false; // Dip-Switch 1 bool dip2 = false; // Dip-Switch 2 bool dip3 = false; // Dip-Switch 3 bool dip4 = false; // Dip-Switch 4  reader.sendProtocol((byte)0x74);  in1 = reader.getBooleanData(FEDM_ISC_TMP_INP_STATE_IN1); in2 = reader.getBooleanData(FEDM_ISC_TMP_INP_STATE_IN2); dip1 = reader.getBooleanData(FEDM_ISC_TMP_INP_STATE_DIP1); dip2 = reader.getBooleanData(FEDM_ISC_TMP_INP_STATE_DIP2); dip3 = reader.getBooleanData(FEDM_ISC_TMP_INP_STATE_DIP3); dip4 = reader.getBooleanData(FEDM_ISC_TMP_INP_STATE_DIP4); </pre>
[0x75] Adjust Antenna	<pre> int antValue = 0; // Antenna voltage  reader.sendProtocol((byte)0x75);  antValue = reader.getIntegerData(FEDM_ISCM_TMP_ANTENNA_VALUE); </pre>
[0x80] Read Configuration und [0x81] Write Configuration	<pre> // The sample shows the read and write back function of one block in the reader configuration  byte cfgAdr = 2; // Address of the configuration block bool eeProm = true; // Configuration datas from/for reader's EEPROM byte busAddress; // Bus address of the ISC.LR-reader from block 2  reader.setData(FEDM_ISC_TMP_READ_CFG, (byte)0x00); // reset all reader.setData(FEDM_ISC_TMP_READ_CFG_ADR, cfgAdr); // set address reader.setData(FEDM_ISC_TMP_READ_CFG_LOC, eeProm); // set Memory location to EEPROM  reader.sendProtocol((byte)0x80); // read configuration data  busAddress = reader.getConfigParaAsByte(ReaderConfig.HostInterface.Serial.BusAddress);  reader.setData(FEDM_ISC_TMP_WRITE_CFG, (byte)0x00); // reset all reader.setData(FEDM_ISC_TMP_WRITE_CFG_ADR, cfgAdr); // set address reader.setData(FEDM_ISC_TMP_WRITE_CFG_LOC, eeProm); // set Memory location to EEPROM  reader.sendProtocol((byte)0x81); // write back configuration data </pre>

[Control Byte] Protocol	Example
[0x82] Save Configuration	<pre> reader.setData(FEDM_ISC_TMP_SAVE_CFG, (byte)0x00); // reset all reader.setData(FEDM_ISC_TMP_SAVE_CFG_ADR, (byte)0x00); // set address reader.setData(FEDM_ISC_TMP_SAVE_CFG_MODE, true); // save all blocks  reader.sendProtocol((byte)0x82); // Save configuration data from RAM into EEPROM </pre>
[0x83] Set Default Configuration	<pre> reader.setData(FEDM_ISC_TMP_RESET_CFG, (byte)0x00); // reset all reader.setData(FEDM_ISC_TMP_RESET_CFG_ADR, (byte)0x02); // set address reader.setData(FEDM_ISC_TMP_RESET_CFG_LOC, false); // choose RAM reader.setData(FEDM_ISC_TMP_RESET_CFG_MODE, false); // set default only block 2  reader.sendProtocol((byte)0x83); // Set configuration data from block 2 in RAM to default </pre>
[0x85] Set System Timer	<pre> reader.setData(FEDM_ISCLR_TMP_TIME_H, (int)16); // 16 hours reader.setData(FEDM_ISCLR_TMP_TIME_M, (int)20); // 20 minutes reader.setData(FEDM_ISCLR_TMP_TIME_MS, (int)2000); // 2000 milliseconds  reader.sendProtocol((byte)0x85); // set Timer </pre>
[0x86] Get System Timer	<pre> int hour = 0; // hours int minute = 0; // minutes int milliSec = 0; // milliseconds  reader.sendProtocol((byte)0x86); // read timer  hour = reader.getIntegerData(FEDM_ISCLR_TMP_TIME_H); // take over hours minute = reader.getIntegerData(FEDM_ISCLR_TMP_TIME_M); // take over minutes milliSec = reader.getIntegerData(FEDM_ISCLR_TMP_TIME_MS); // take over milliseconds </pre>
[0x87] Set System Date	<pre> reader.setData(FEDM_ISC_TMP_DATE_CENTURY, (byte)20); // 20<sup>th</sup> century reader.setData(FEDM_ISC_TMP_DATE_YEAR, (byte)4); // year 04 reader.setData(FEDM_ISC_TMP_DATE_MONTH, (byte)9); // September reader.setData(FEDM_ISC_TMP_DATE_DAY, (byte)15); // 15th September reader.setData(FEDM_ISC_TMP_DATE_TIMEZONE, (byte)0); // actually unused reader.setData(FEDM_ISC_TMP_DATE_HOUR, (byte)12); // hour reader.setData(FEDM_ISC_TMP_DATE_MINUTE, (byte)00); // minute reader.setData(FEDM_ISC_TMP_DATE_MILLISECOND, (int)0); // milliseconds (with seconds)  reader.sendProtocol((byte)0x87); // set date and time </pre>
[0x88] Get System Date	<pre> byte century = 0; byte year = 0; byte month = 0; byte day = 0; byte timezone = 0; byte hour = 0; byte minute = 0; int milliSec = 0;  reader.sendProtocol((byte)0x88); // read date and time  century = reader.getByteData(FEDM_ISC_TMP_DATE_CENTURY); year = reader.getByteData(FEDM_ISC_TMP_DATE_YEAR); month = reader.getByteData(FEDM_ISC_TMP_DATE_MONTH); day = reader.getByteData(FEDM_ISC_TMP_DATE_DAY); timezone = reader.getByteData(FEDM_ISC_TMP_DATE_TIMEZONE); hour = reader.getByteData(FEDM_ISC_TMP_DATE_HOUR); minute = reader.getByteData(FEDM_ISC_TMP_DATE_MINUTE); milliSec = reader.getIntData(FEDM_ISC_TMP_DATE_MILLISECOND); </pre>

[Control Byte] Protocol	Example
[0xA0] Reader Login	<pre>byte[] passWord = new byte[4];  passWord[0] = 0x00; passWord[1] = 0x00; passWord[2] = 0x00; passWord[3] = 0x00;      // Password  reader.setData(FEDM_ISCLR_TMP_READER_PW, passWord);      // set Password  reader.sendProtocol((byte)0xA0);      // send Password to the reader</pre>
[0xA2] Write Mifare Keys	<pre>Byte key[] = new byte[6];  // take the Mifare-Key e.g. from an input field  reader.setData(FEDM_ISC_TMP_ISO14443A_KEY_TYPE, (byte)0); reader.setData(FEDM_ISC_TMP_ISO14443A_KEY_ADR, (byte)0); reader.setData(FEDM_ISC_TMP_ISO14443A_KEY, key);  reader.sendProtocol((byte)0xB0);      // send Mifare-Key to the reader</pre>
[0xAD] Write Reader Authent Key	<pre>Byte key[] = new byte[32];  // take the Authent-Key e.g. from an input field (32 chars containing hex-characters 0..9, A..F)  reader.setData(FEDM_ISC_TMP_0xAD_KEY_TYPE, (byte)2); // AES256 reader.setData(FEDM_ISC_TMP_0xAD_KEY_LEN, (byte)32); reader.setData(FEDM_ISC_TMP_0xAD_KEY, key);  reader.sendProtocol((byte)0xAD);      // write Authent-Key into the reader</pre>
[0xB0] ISO Mandatory and Optional Commands	<pre>// the sample shows the [0x01] Inventory  reader.setData(FEDM_ISC_TMP_B0_CMD, (byte)0x01);      // Inventory reader.setData(FEDM_ISC_TMP_B0_MODE, (byte)0x00);      // no More-Flag  reader.sendProtocol((byte)0xB0);  // the Inventory-data are in the m_ISOTable. Samples for data access in <a href="#">5.8.1. Examples for using the table for ISO-Host Mode</a></pre>
[0xB1] ISO15693 Customer and Proprietary Commands  (only for Transponder from Philips Electronics N.V.)	<pre>// the sample shows the [0xA2] Set EAS  // all others correspond to the 0xB1 commands  String snr = new String();      // for Serial number byte isoError = 0;      // for ISO-Error code  reader.setData(FEDM_ISC_TMP_B1_CMD, (byte)0xA2);      // Set EAS reader.setData(FEDM_ISC_TMP_B1_MFR, (byte) ISO_MFR_PHILIPS);      // Manufacturer reader.setData(FEDM_ISC_TMP_B1_MODE, (byte) ISO_MODE_ADR);      // addressed  // ... Serial number e.g. take from text field and store in sSnr reader.setData(FEDM_ISC_TMP_B1_REQ_UID, snr);  int status = reader.sendProtocol((byte)0xB1);  if(status == 0x95)  {      // take ISO-Error code     isoError = reader.getIntegerData(FEDM_ISC_TMP_B1_ISO_ERROR);  }</pre>

[Control Byte] Protocol	Example
<p>[0xB2] ISO14443 Special Commands</p> <p>[0x2B] ISO14443-4 Transponder Info</p>	<pre> byte FSCI = 0; byte FWI = 0; byte DSI = 0; byte DRI = 0; byte Nad = 0; byte Cid = 0;  reader.setData(FEDM_ISC_TMP_B2_CMD, (byte)0x2B); // ISO14443-4 Transponder Info  int status = reader.sendProtocol(0xB2); // transponder must previously selected with // [0x25] Select  if(status == 0x00) {     // get the table index of the selected transponder     int idx = reader.findTableIndex(0, ISO_TABLE, DATA_IS_SELECTED, true);     if(idx &gt;= 0)     {         // get transponder data         FSCI = reader.getTableData(idx, ISO_TABLE, DATA_FSCI)         FWI = reader.getTableData(idx, ISO_TABLE, DATA_FWI)         DSI = reader.getTableData(idx, ISO_TABLE, DATA_DSI)         DRI = reader.getTableData(idx, ISO_TABLE, DATA_DRI)         NAD = reader.getTableData(idx, ISO_TABLE, DATA_NAD)         CID = reader.getTableData(idx, ISO_TABLE, DATA_CID)     } } </pre>
<p>[0xB2] ISO14443 Special Commands</p> <p>[0xB0] Authent Mifare</p>	<pre> byte dbAddress = 0; // Address of the first data block byte keyType = 0; // Keytype for authentifikation 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((byte)0xB2); </pre>
<p>[0xB2] ISO14443 Special Commands</p> <p>[0xB1] Authent my-d</p>	<pre> byte keyAdrTag = 5; // Address of the Keys on the Transponder byte keyAdrSam = 2; // Address of the Keys in the Authentifikation module byte cntAdr = 3; // Address of the Authentifikation counter byte authSeq = 0; // Authentifikation sequence  reader.setData(FEDM_ISC_TMP_B2_CMD, (byte)0xB1); // Authent my-d reader.setData(FEDM_ISC_TMP_B2_MODE, (byte) FEDM_ISC_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); </pre>

[Control Byte] Protocol	Example
	<pre>reader.sendProtocol((byte)0xB2);</pre>
<p>[0xB2] ISO14443 Special Commands</p> <p>[0x30] Mifare Value Commands</p>	<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>



## 5.8. 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 *getXXXTableData*<sup>6</sup>, *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 *getXXXTableData* is also accomplished using the methods *getData* and *setData* 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 getIntegerTableData( 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](#)).

---

<sup>6</sup> XXX stands for Boolean, Byte, Integer, Long, String and represents the data types of the return value.

### 5.8.1. Examples for using the table for ISO-Host Mode

#### 5.8.1.1. 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 32 byte)
reader.setData(FEDM_ISC_TMP_B0_REQ_UID, uid);
reader.setData(FEDM_ISC_TMP_B0_REQ_UID_LEN, uidLen);    // number of byte in UID

reader.setData(FEDM_ISC_TMP_B0_CMD, (byte)0x23);        // Command Read Multiple Blocks
reader.setData(FEDM_ISC_TMP_B0_MODE, (byte)0x00);        // clear mode byte
reader.setData(FEDM_ISC_TMP_B0_MODE_ADR, (byte)0x01);    // addressed mode
reader.setData(FEDM_ISC_TMP_B0_MODE_UID_LF, true);        // UID_LF flag
reader.setData(FEDM_ISC_TMP_B0_REQ_DBN, (byte)0x01);    // request one data block
reader.setData(FEDM_ISC_TMP_B0_REQ_DB_ADR, dbAdr);        // set data block address

reader.sendProtocol(0xB0);    // communication wit reader/transponder
```

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

[Control Byte] Protocol	Example
[0x01] Inventory  <b>for:</b>  <u>HF-Transponder</u> - Philips I-CODE1 - Texas Instruments Tag-it HF - ISO15693 - ISO14443A - ISO14443B - EPC (Electronic Product Code) - Philips I-CODE UID - Innovision Jewel - EPC Class1 Gen2 HF  <u>UHF-Transponder</u> - ISO18000-6-A - ISO18000-6-B - EM4222 - EPC Class0/0+ - EPC Class1 Gen1 - EPC Class1 Gen2	<pre>byte trType = 0;    // for Transponder type String snr = new String();    // for Serial number (also EPC) String header = new String();    // for EPC Header String domain = new String();    // for EPC DomainManager-Field String object = new String();    // for EPC ObjektClass-Field String epc = new String();    // for EPC ("Header.DomainManager.ObjectClass.Serialnumber")  reader.setData(FEDM_ISC_TMP_B0_CMD, (byte)0x01);    // Command Inventory reader.setData(FEDM_ISC_TMP_B0_MODE, (byte)0x00);    // no More-Flag // set table length to 0 and delete the content of the table completely reader.deleteTable(ISO_TABLE);  reader.sendProtocol((byte)0xB0);    // Communication with reader/transponder  // All transponder data are in the table for(int cnt=0; cnt&lt; reader.getTableLength(ISO_TABLE); ++cnt) {     // get transponder typ     trType = reader.getByteTableData(cnt, ISO_TABLE, DATA_TRTYPE);     switch(trType)     {         case 0x00: // Philips I-CODE1         case 0x01: // Texas Instruments Tag-it HF         case 0x03: // ISO15693</pre>

[Control Byte] Protocol	Example
	<pre> case 0x04: // ISO14443A case 0x05: // ISO14443B case 0x07: // I-Code UID case 0x08: // Innovision Jewel case 0x09: // EPC Class1 Gen2 HF case 0x81: // ISO18000-6-B case 0x83: // EM4222 case 0x84: // EPC Class1 Gen2 case 0x88: // EPC Class0/0+ case 0x89: // EPC Class1 Gen1     // get serial number as String     snr = reader.getStringTableData(cnt, ISO_TABLE, DATA_SNR);     break; case 0x06: // EPC (Electronic Product Code)     // get EPC-Fields     header = reader.getStringTableData(cnt, ISO_TABLE, DATA_EPC_HEADER);     domain = reader.getStringTableData(cnt, ISO_TABLE, DATA_EPC_DOMAIN);     object = reader.getStringTableData (cnt, ISO_TABLE, DATA_EPC_OBJECT);     snr = reader.getStringTableData (cnt, ISO_TABLE, DATA_EPC_SNR);     // or get EPC-Field as complete String     epc = reader.getStringTableData (cnt, ISO_TABLE, DATA_EPC);     break;     }     }</pre>
[0x02] Stay Quiet	<pre> String snr = new String;           // for Serial number  // ... take serial number e.g. from text field and store it in snr  // set Serial number for Addressed Mode reader.setData(FEDM_ISC_TMP_B0_REQ_UID, snr);  reader.setData(FEDM_ISC_TMP_B0_CMD, (byte)0x02);           // Command Stay Quiet 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.sendProtocol((byte)0xB0); // Communication with reader/transponder</pre>
[0x22] Lock Multiple Blocks	<pre> /* <u>Attention</u>: with this ISO Command all data blocks will be locked irreversible!!  String snr = new String;           // for Serial number  // ... take serial number e.g. from text field and store it in snr  // determine table index of the serial number int idx = reader.findTableIndex(0, ISO_TABLE, DATA_SNR, snr); if(idx &lt; 0)     return;  // set serial number for Addressed Mode reader.setData(FEDM_ISC_TMP_B0_REQ_UID, snr);  reader.setData(FEDM_ISC_TMP_B0_CMD, (byte)0x22);           // Command Lock Multiple Blocks 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_DBN, (byte)0x01);       // lock one Data block reader.setData(FEDM_ISC_TMP_B0_REQ_DB_ADR, (byte)0x00);     // set Data block-Address  reader.sendProtocol((byte)0xB0); // Communication with reader/transponder</pre>
[0x23] Read Multiple Blocks	<pre> byte[] dataBlock;           // Buffer for one data block</pre>

[Control Byte] Protocol	Example
(standard address mode)	<pre> byte dbAddress = 5;           // Data block-address 5 String snr = new String();    // for serial number  // ... Serial number e.g. take from text field  // set serial number for addressed mode reader.setData(FEDM_ISC_TMP_B0_REQ_UID, snr);  reader.setData(FEDM_ISC_TMP_B0_CMD, (byte)0x23);    // Command Read Multiple Blocks 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_DBN, (byte)0x01);    // read one Data block reader.setData(FEDM_ISC_TMP_B0_REQ_DB_ADR, dbAddress); // set Data block-Address  reader.sendProtocol((byte)0xB0); // Communication with reader/transponder  // All Transponder data are content in the table  // first determine the table index of the serial number int idx = reader.findTableIndex(0, ISO_TABLE, DATA_SNR, snr); if(idx &lt; 0)     return;  // take the size of the data blocks (Block size) byte blockSize = reader.getBytesTableData(idx, ISO_TABLE, DATA_BLOCKSIZE); // ... do something with the block size  // take a data block (data block contents only the block size data byte) dataBlock = reader.getBytesArrayTableData(idx, ISO_TABLE, DATA_RxDB, dbAddress); // ... do something with the data block </pre>
[0x23] Read Multiple Blocks (extended address mode)	<pre> byte[] dataBlock;           // buffer for one data block byte 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 (&gt; 8 Byte acceptable) for addressed mode reader.setData(FEDM_ISC_TMP_B0_REQ_UID, snr); reader.setData(FEDM_ISC_TMP_B0_REQ_UID_LEN, snr.length()/2 ); // length of UID in bytes reader.setData(FEDM_ISC_TMP_B0_CMD, (byte)0x23);    // Command Read Multiple Blocks reader.setData(FEDM_ISC_TMP_B0_MODE, (byte)0x00);    // clear mode byte reader.setData(FEDM_ISC_TMP_B0_MODE_ADR, (byte)0x01);    // Addr. Mode reader.setData(FEDM_ISC_TMP_B0_MODE_EXT_ADR, true); // extended addressed mode reader.setData(FEDM_ISC_TMP_B0_MODE_UID_LF, true);    // length of UID != 8 reader.setData(FEDM_ISC_TMP_B0_BANK, (UCHAR)0x00);    // clear bank nyte reader.setData(FEDM_ISC_TMP_B0_BANK_BANK_NR, (UCHAR)0x03); // bank User Memory reader.setData(FEDM_ISC_TMP_B0_BANK_ACCESS_FLAG, true); // with access password reader.setData(FEDM_ISC_TMP_B0_ACCESS_PW_LENGTH, (byte)sPw.length()/2); // len in bytes reader.setData(FEDM_ISC_TMP_B0_ACCESS_PW, sPw);        // password reader.setData(FEDM_ISC_TMP_B0_REQ_DB_ADR_EXT, dbAddress); // datablock address reader.setData(FEDM_ISC_TMP_B0_REQ_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); </pre>

[Control Byte] Protocol	Example
	<pre> if(idx &lt; 0)     return; // take the size of the data blocks (Block size) byte blockSize; reader.<b>getTableData</b>(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.<b>getTableData</b>(idx, ISO_TABLE, DATA_RxDB, dbAddress, out dataBlock); // ... do something with the data block </pre>
<p>[0x24] Write Multiple Blocks (standard address mode)</p>	<pre> /* the example shows the [0x24] Write Multiple Blocks. 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 <b>getBooleanTableData</b>(..., 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 ucDBAdr = 5;           // Data block-address 5 String snr = new String();  // for serial number  // ... Serial number e.g. take from Text field and store it in snr  // ... Daten block e.g. take from Text field and store it in dataBlock  // determine table index of the serial-number int idx = reader.<b>findTableIndex</b>(0, ISO_TABLE, DATA_SNR, snr); if(idx &lt; 0)     return;  // set serial-number for Addressed Mode reader.<b>setData</b>(FEDM_ISC_TMP_B0_REQ_UID, snr);  reader.<b>setData</b>(FEDM_ISC_TMP_B0_CMD, (byte)0x24);           // Command Read Multiple Blocks reader.<b>setData</b>(FEDM_ISC_TMP_B0_MODE, (byte)0x00);         // Mode-Byte reset reader.<b>setData</b>(FEDM_ISC_TMP_B0_MODE_ADR, (byte)0x01);    // Addressed Mode reader.<b>setData</b>(FEDM_ISC_TMP_B0_REQ_DBN, (byte)0x01);     // write one data block reader.<b>setData</b>(FEDM_ISC_TMP_B0_REQ_DB_ADR, dbAddress);   // set data block-address  reader.<b>setTableData</b>(idx, ISO_TABLE, DATA_BLOCK_SIZE, (byte)0x08); // set block size to 8  // write one data block (with blocksize of 8 bytes!) in the table reader.<b>setTableData</b>(idx, ISO_TABLE, DATA_TxDB, ucDBAdr, dataBlock);  reader.<b>sendProtocol</b>((byte)0xB0); // Communication with reader/transponder </pre>

[Control Byte] Protocol	Example
<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 byte 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 &lt; 0)     return;  // set serial-number for Addressed Mode reader.setData(FEDM_ISC_TMP_B0_REQ_UID, snr); reader.setData(FEDM_ISC_TMP_B0_REQ_UID_LEN, snr.length()/2 ); // length of UID in byte reader.setData(FEDM_ISC_TMP_B0_CMD, (byte)0x24); // Command Read Multiple Blocks reader.setData(FEDM_ISC_TMP_B0_MODE, (byte)0x00); // clear mode byte reader.setData(FEDM_ISC_TMP_B0_MODE_ADR, (byte)0x01); // addressed mode reader.setData(FEDM_ISC_TMP_B0_MODE_EXT_ADR, true); // extended addressed mode reader.setData(FEDM_ISC_TMP_B0_MODE_UID_LF, true); // length of UID != 8 reader.setData(FEDM_ISC_TMP_B0_BANK, (UCHAR)0x00); // clear bank nyte reader.setData(FEDM_ISC_TMP_B0_BANK_BANK_NR, (UCHAR)0x03); // bank User Memory reader.setData(FEDM_ISC_TMP_B0_BANK_ACCESS_FLAG, true); // with access password reader.setData(FEDM_ISC_TMP_B0_ACCESS_PW_LENGTH, (byte)sPw.length()/2); //Len in bytes reader.setData(FEDM_ISC_TMP_B0_ACCESS_PW, sPw); // password reader.setData(FEDM_ISC_TMP_B0_REQ_DB_ADR_EXT, dbAddress); // datablock address reader.setData(FEDM_ISC_TMP_B0_REQ_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>
<p>[0x25] Select</p>	<pre> String snr = new String; // for Serial-number  // ... Serial number e.g. take from Text field and store it in snr  // set Serial-number for Addressed Mode reader.setData(FEDM_ISC_TMP_B0_REQ_UID, snr);  reader.setData(FEDM_ISC_TMP_B0_CMD, (byte)0x25); // Command Select 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.sendProtocol((byte)0xB0); // Communication with reader/transponder </pre>
<p>[0x25] Select</p>	<pre> String snr = new String; // for Serial-number </pre>

[Control Byte] Protocol	Example
mit Option Card Information für ISO14443 Transponder	<pre> 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(FEDM_ISC_TMP_B0_REQ_UID, snr);  reader.setData(FEDM_ISC_TMP_B0_CMD, (byte)0x25);           // Command Select 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_MODE_CINF, true);          // CINF-Flag  reader.sendProtocol((byte)0xB0); // Communication with reader/transponder  // the Format byte is stored in TMPDATA_MEM format = reader.getData(FEDM_ISC_TMP_B0_RSP_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. system manual // cardInfo = reader. getByteArrayData(2048, length, TMPDATA_MEM); </pre>
[0x26] Reset to Ready	<pre> String snr = new String;           // for serial-number  // ... Serial number e.g. take from Text field and store it in snr  // set serial-number for Addressed Mode reader.setData(FEDM_ISC_TMP_B0_REQ_UID, snr);  reader.setData(FEDM_ISC_TMP_B0_CMD, (byte)0x26);           // Command Reset to Ready 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.sendProtocol((byte)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 entry field and store it in afi  // set serial-number for Addressed Mode reader.setData(FEDM_ISC_TMP_B0_REQ_UID, snr);  // determine table index of the serial-number int idx = reader.findTableIndex(0, ISO_TABLE, DATA_SNR, snr); if(idx &lt; 0)     return;  // write AFI in table reader.setTableData(idx, ISO_TABLE, DATA_AFI, afi);  reader.setData(FEDM_ISC_TMP_B0_CMD, (byte)0x27);           // Command Write AFI 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.sendProtocol((byte)0xB0); // Communication with reader/transponder </pre>
[0x28] Lock AFI	<pre> String snr = new String;           // for Serial number  // ... Serial number e.g. take from text field and store it in snr </pre>



[Control Byte] Protocol	Example
	<pre> // set serial number for Addressed Mode reader.setData(FEDM_ISC_TMP_B0_REQ_UID, snr);  reader.setData(FEDM_ISC_TMP_B0_CMD, (byte)0x28);           // Command Lock AFI 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.sendProtocol((byte)0xB0); // Communication with reader/transponder </pre>
[0x29] Write DSFID	<pre> String snr = new String;           // 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(FEDM_ISC_TMP_B0_REQ_UID, snr);  // determine table index of the serial number int idx = reader.findTableIndex(0, ISO_TABLE, DATA_SNR, snr); if(idx &lt; 0)     return;  // write DSFID in table reader.setTableData(idx, ISO_TABLE, DATA_DSFID, dsfid);  reader.setData(FEDM_ISC_TMP_B0_CMD, (byte)0x29);           // Command Write DSFID 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.sendProtocol((byte)0xB0); // Communication with reader/transponder </pre>
[0x2A] Lock DSFID	<pre> String snr = new String;           // for Serial number  // ... Serial number e.g. take from text field and store it in snr  // set serial number for Addressed Mode reader.setData(FEDM_ISC_TMP_B0_REQ_UID, snr);  reader.setData(FEDM_ISC_TMP_B0_CMD, (byte)0x2A);           // Command Lock DSFID 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.sendProtocol((byte)0xB0); // Communication with reader/transponder </pre>
[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(FEDM_ISC_TMP_B0_REQ_UID, snr);  reader.setData(FEDM_ISC_TMP_B0_CMD, (byte)0x2B); // Command Get System Information 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.sendProtocol((byte)0xB0); // Communication with reader/transponder  // All transponder data content in the table </pre>



[Control Byte] Protocol	Example
	<pre> // first determine table index of the serial number int idx = reader.findTableIndex(0, ISO_TABLE, DATA_SNR, snr); if(idx &lt; 0)     return;  // get AFI afi = reader.getBytesTableData(idx, ISO_TABLE, DATA_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;      // for Serial number  // ... Serial number e.g. take from text field and store it in snr  // set serial number for Addressed Mode reader.setData(FEDM_ISC_TMP_B0_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 1. Data block-address  reader.setData(FEDM_ISC_TMP_B0_CMD, (byte)0x2C); // Command Get Multiple Block // Security Status  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.sendProtocol((byte)0xB0); // Communication with reader/transponder  // All transponder data content in the table  // first determine table index of the serial number int idx = reader.findTableIndex(0, ISO_TABLE, DATA_SNR, snr); if(idx &lt; 0)     return;  // get the security status from block 0..4 for(int cnt=0; cnt&lt;5; ++cnt) {     secStatus = reader.getBytesTableData(idx, ISO_TABLE, DATA_SEC_STATUS, cnt);     // ... do something with secStatus } </pre>
[0xA0] Read Config Block	<pre> byte[] configBlock; // buffer for one Data block (Block size 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 reader.setData(FEDM_ISC_TMP_B0_REQ_UID, snr);  reader.setData(FEDM_ISC_TMP_B0_CMD, (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((byte)0xB0); // Communication with reader/transponder  // All transponder data content in the table  // first determine table index of the serial number </pre>

[Control Byte] Protocol	Example
	<pre> int idx = reader.findTableIndex(0, ISO_TABLE, DATA_SNR, snr); if(idx &lt; 0)     return;  // get the data block configBlock = reader.getBytesTableData(idx, ISO_TABLE, DATA_RxCB, cbAddress); // ... do something with the data block </pre>
[0xA1] Write Config Block	<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 (Block size 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  // ... take data block e.g. take it from a text field and store it in the configBlock  // determine the table index of the serial number int idx = reader.findTableIndex(0, ISO_TABLE, DATA_SNR, snr); if(idx &lt; 0)     return;  // set serial number for Addressed Mode reader.setData(FEDM_ISC_TMP_B0_REQ_UID, snr);  reader.setData(FEDM_ISC_TMP_B0_CMD, (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);       // Addressed Mode reader.setData(FEDM_ISC_TMP_B0_REQ_CB_ADR, cbAddress); // set data block-address  // write a data block in the table reader.setTableData(idx, ISO_TABLE, DATA_TxCB, cbAddress, configBlock);  reader.sendProtocol((byte)0xB0); // Communication with reader/transponder </pre>

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

[Control byte] protocol	Example <sup>7</sup>
<p>[0x18] Kill</p> <p>for UHF-Transponder:</p> <ul style="list-style-type: none"> <li>- EPC Class1 Gen1</li> <li>- EPC Class1 Gen2</li> </ul>	<pre>// Attention: with this command transponders are destroyed irretrievably!  String epc;           // for EPC String pw;            // for Kill Password byte epcLen = 0;      // length of EPC in byte byte pwLen = 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, epc);  // set EPC for addressed mode reader.setData(FEDM_ISC_TMP_B3_REQ_EPC, epc); reader.setData(FEDM_ISC_TMP_B3_REQ_EPC_LEN, epcLen); // 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, pwLen); // length of Kill Password reader.setData(FEDM_ISC_TMP_B3_KILL_PW, pw);          // 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"> <li>- EPC Class1 Gen1</li> <li>- EPC Class1 Gen2</li> </ul>	<pre>// Attention: with this ISO Command all data blocks will be locked irretrievably!  string epc;           // for EPC string lockData;      // for Lock data string pw;            // for Access Password byte epcLen = 0;      // length of EPC in byte byte trType = 0;      // transponder type byte lockDataLen = 0; // length of Lock Data in byte byte pwLen = 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 // ... Access 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 trType = reader.getByteTableData(idx, ISO_TABLE, DATA_TRTYPE);  // set EPC for addressed mode reader.setData(FEDM_ISC_TMP_B3_REQ_EPC, epc); reader.setData(FEDM_ISC_TMP_B3_REQ_EPC_LEN, epcLen); // 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</pre>

<sup>7</sup> all examples in C#

[Control byte] protocol	Example <sup>7</sup>
	<pre> reader.setData(FEDM_ISC_TMP_B3_REQ_TR_TYPE, trType); // transponder type reader.setData(FEDM_ISC_TMP_B3_LOCK_DATA_LENGTH, lockDataLen); // length of Lock // data reader.setData(FEDM_ISC_TMP_B3_LOCK_DATA, sLockData); // Lock data reader.setData(FEDM_ISC_TMP_B3_ACCESS_PW_LENGTH, pwLen); // length of Access // Password  if(pwLen &gt; 0)     reader.setData(FEDM_ISC_TMP_B3_ACCESS_PW, sw); // Access Password  reader.sendProtocol(0xB3); // kommunikation 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.  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[][] db; // buffer for Data (1. dimension for block number, 2. dimension für data) string epc; // for EPC string pw; // for optional Access Password byte epcLen = 0; // length of EPC in byte byte pwLen = 0; // length of optional Access Password  // ... EPC e.g. take from Text field and store it in epc // ... Access Password e.g. take from text field and store it in pw, dito with the length // ... data block e.g. take from Text field and store it in db  // determine table index of the EPC int idx = reader.findTableIndex(0, ISO_TABLE, DATA_SNR, epc);  // set EPC for addressed mode reader.setData(FEDM_ISC_TMP_B3_REQ_UID, epc); reader.setData(FEDM_ISC_TMP_B3_REQ_EPC_LEN, epcLen); // 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 adr=0; adr&lt;6; ++adr)     reader.setTableData(idx, ISO_TABLE, DATA_TxDB, adr, db[adr]);  reader.sendProtocol(0xB3); // communication with Reader/Transponder </pre>

## 5.8.2. Examples for using the table for Buffered Read Mode

[Control Byte] Protocol	Example
[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 Protocol byte[] dataBlock;           // Buffer for a data block FelscReaderTime time = 0;    // for Timer-value String snr;                 // for serial number boolean snrFlag = false;     // Flag for serial number in dataset boolean dbFlag = false;      // Flag for data block in Dataset boolean timerFlag = false;   // Flag for Timer in Datenset FedmBrmTableItem item;       // a table entry with data for a transponder  reader.setData(FEDM_ISCLR_TMP_BRM_SETS, dataSets);  reader.sendProtocol((byte)0x21); // read blocks from transponder with Buffered Read Mode  snrFlag = reader.getBooleanData(FEDM_ISCLR_TMP_BRM_TRDATA_SNR); dbFlag = reader.getBooleanData(FEDM_ISCLR_TMP_BRM_TRDATA_DB); timerFlag = reader.getBooleanData(FEDM_ISCLR_TMP_BRM_TRDATA_TIME); recSets = reader.getBytesData(FEDM_ISCLR_TMP_BRM_RECSETS);  // All transponder data content in the table for(int cnt=0; cnt&lt; reader.getTableLength(BRM_TABLE); cnt++) {     item = (FedmBrmTableItem) reader.getTableItem(cnt, BRM_TABLE);     if(snrFlag) // get serial number         snr = item.getStringData(DATA_SNR);      if(dbFlag) // get data block 1         dataBlock = item.getBytesData(DATA_RxDB);      if(timerFlag) // get Timer-value         time = item.getReaderTime(); } </pre>
[0x22] Read Buffer	<pre> // this sample shows the reading of Data sets with serial number, antenna number and Timer-value byte[] dataBlock;           // Puffer für einen Datenblock  int dataSets = 1;           // Number requested data sets int recSets = 0;           // Number of received data sets FelscReaderTime time = 0;   // for date and time value String snr;                 // for serial number String db;                  // for data blocks byte blockSize = 0;         // for blocksize int dbn = 0;                // for number of data blocks byte antennaNumber = 0;     // for antenna number byte input = 0;             // for input byte byte state = 0;             // for status byte boolean snrFlag = false;     // flag (in TR-DATA1) for serial number in dataset boolean dbFlag = false;     // flag (in TR-DATA1) for data block in dataset boolean antFlag = false;    // flag (in TR-DATA1) for antenna number in dataset boolean timeFlag = false;   // flag (in TR-DATA1) for time in dataset boolean dateFlag = false;   // flag (in TR-DATA1) for date in dataset </pre>

[Control Byte] Protocol	Example
	<pre> boolean extFlag = 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 boolean inputFlag = 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(FEDM_ISC_TMP_ADV_BRM_SETS, dataSets);  reader.sendProtocol((byte)0x22); // read data from transponder with Buffered Read Mode  snrFlag = reader.getBooleanData(FEDM_ISC_TMP_ADV_BRM_TRDATA1_SNR); antFlag = reader.getBooleanData(FEDM_ISC_TMP_ADV_BRM_TRDATA1_ANT); timeFlag = reader.getBooleanData(FEDM_ISC_TMP_ADV_BRM_TRDATA1_TIME); dateFlag = reader.getBooleanData(FEDM_ISC_TMP_ADV_BRM_TRDATA1_DATE); extFlag = reader.getBooleanData(FEDM_ISC_TMP_ADV_BRM_TRDATA1_EXT); inputFlag = reader.getBooleanData(FEDM_ISC_TMP_ADV_BRM_TRDATA2_INPUT); recSets = reader.getIntegerData(FEDM_ISC_TMP_ADV_BRM_RECSETS);  // All transponder data content in the table for(int cnt=0; cnt&lt; reader.getTableLength(BRM_TABLE); cnt++) {     item = (FedmBrmTableItem) reader.getTableItem(cnt, BRM_TABLE);     if(snrFlag) // get serial number         snr = item.getStringData(DATA_SNR);      if(db) // get all data blocks     {         // get number of data blocks         dbn = item.getIntegerData(iCnt, DATA_DBN);         // get the blocksize         blockSize = item.getData(iCnt, DATA_BLOCK_SIZE);         // get data blocks         for(int i=0; i&lt;dbn; ++i)         {             db = item.getByteArrayData(iCnt, DATA_RxDB, i);             // do anything with the data blocks         }     }      if(antFlag) // get antenna number         antennaNumber = item.getByteData(DATA_ANT_NR);      if(timerFlag    dateFlag) // get date and/or time value         time = item.getReaderTime();      if(extFlag &amp;&amp; inputFlag) // get input and status byte     {         input = item.getByteData(DATA_INPUT);         state = item.getByteData(DATA_STATE);     } } </pre>

## 5.9. Example for using the method sendSAMCommand

---

The method *sendSAMCommand* of the reader class **FedmIscReader** executes an asynchronous communication with the connected OBID® *classic-pro* 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.

```
import de.feig.*;
```

For demonstration a test objekt *myClass* is defined which implements the interface *FedmTaskListener*. The constructor gets a reader object *reader*:

```
MyClass myClass = new MyClass(reader);
```

```
public class MyClass implements FedmTaskListener
{
    FedmIscReader reader;

    MyClass(FedmIscReader reader)
    {
        this.reader = reader;
    }

    public void send()
    {
        byte[] data = new byte[2];
        // Activate T=0
        data[0] = 0x01;
        data[1] = 0x01;

        // SAM-Slot 1, Timeout = 1s (10*100ms)
        // execute asynchronous communication
        reader.sendSAMCommand(this, 1, data, 10);
    }

    public void onNewSAMResponse(int error, byte[] responseData)
    {
        //if error = 0, responseData can contain data
    }
}
```

---

## 5.10. Example for using the method sendTclApdu

---

The method *sendTclApdu* of the reader class **FedmIscReader** executes an asynchronous communication with the connected OBID® *classic-pro* 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.

```
import de.feig.*;
```

For demonstration a test objekt *myClass* is defined which implements the interface *FedmTaskListener*. The constructor gets a reader object *reader*:

```
MyClass myClass = new MyClass(reader);

public class MyClass implements FedmTaskListener
{
    FedmIscReader reader;
    FedmCprApdu apdu =new FedmCprApdu(this);  // APDU objekt

    MyClass(FedmIscReader reader)
    {
        this.reader = reader;
    }

    public void send()
    {
        // prepare APDU objekt
        apdu.setCID(0);
        apdu.setNAD(0);

        // create APDU and add it to APDU objekt
        apdu.setApdu(buildApdu());

        // no use of CID and NAD
        // Timeout is calculated internally
        // execute asynchronous communication
        reader.sendTclApdu(false, false, apdu);
    }

    public void onNewApduResponse(int error)
    {
        //if error = 0, the APDU objekt can contain response data
        if(error)
            return;

        // access to the response data with method:
        // getLastResponseData
    }
}
```



---

## 5.11. Example for using the method `sendCommandQueue`

---

The method `sendCommandQueue` of the reader class **FedmIscReader** executes an asynchronous communication with the connected OBID® *classic-pro* 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.

```
import de.feig.*;
```

For demonstration a test objekt `myClass` is defined which implements the interface `FedmTaskListener`. The constructor gets a reader object `reader`:

```
MyClass myClass = new MyClass(reader);
```

```
public class MyClass implements FedmTaskListener
{
    FedmIscReader reader;
    FedmCprCommandQueue queue =new FedmCprCommandQueue(this); // Queue objekt

    MyClass(FedmIscReader reader)
    {
        this.reader = reader;
    }

    public void send()
    {
        String snr;                // UID (serial number) of Transponder

        // prepare Queue
        queue.clear();
        queue.setMode(0);
        queue.setTimeout(10); // 1s (10*100ms)

        // 1. Command: [0xB0][0x25] Select
        reader.setDataFEDM_ISC_TMP_B0_REQ_UID, snr);           // set UID
        reader.setDataFEDM_ISC_TMP_B0_CMD, (byte)0x25);        // Command Select
        reader.setDataFEDM_ISC_TMP_B0_MODE, (byte)0x00);        // clear mode byte
        reader.setDataFEDM_ISC_TMP_B0_MODE_ADR, (byte)0x01);    // set addressed mode
        reader.addCommand(queue, 0x25);                         // add command to Queue

        // 2. Command: [0xB0][0x23] Read Multiple Blocks
        reader.setDataFEDM_ISC_TMP_B0_CMD, (byte)0x23);        // Read Multiple Blocks
        reader.setDataFEDM_ISC_TMP_B0_MODE, (byte)0x00);        // clear mode byte
        reader.setDataFEDM_ISC_TMP_B0_MODE_ADR, (byte)0x02);    // set selected mode
        reader.setDataFEDM_ISC_TMP_B0_REQ_DBN, (byte)0x01);     // number of data block
        reader.setDataFEDM_ISC_TMP_B0_REQ_DB_ADR, (byte)0x02);  // first data block address
        reader.addCommand(queue, 0x23);                         // add command to Queue

        // execute asynchronous communication
        reader.sendCommandQueue(queue);
    }
}
```

```
public void onNewQueueResponse(int error)
{
    //if error = 0, the Queue objekt can contain response data
    if(error)
        return;

    // access to the response data with methods:
    // getLastCommandStep
    // getLastResponseCommand
    // getLastResponseStatus
    // getLastResponseData
}
}
```

---

## 5.12. Example for communicating with a People Counter

---

```
import de.feig.*;

...

FedmIsedReader reader = new FedmIsedReader();

...

// connecting the reader and execute readReaderInfo() internally
reader.connectTCP("192.168.10.10", 10001);

...

long[] values = null;

// query the map with all People Counter
HashMap<Integer,FedmIsCounter>mapPC= reader.getPeopleCounterMap();

// get People Counter object with busaddress 1
FedmIsCounter pc = mapPC.get(1);

if(pc != null)
{
    try
    {
        // query all counter values
        values = pc.getCounterValues();
    }
    catch(java.lang.Exception e)
    {
        // error handling
    }
}
```

---

### 5.13. Example for use of TagHandler classes

---

```
import de.feig.*;
import de.feig.TagHandler.*;

...

FedmIsedReader reader = new FedmIsedReader();
HashMap<String, FedmIsTagHandler> mapTH = null;
FedmIsTagHandler th = null;
FedmIsTagHandler_Result res = new FedmIsTagHandler_Result();
int back = 0;
int tagDriver = 0;

...

try
{
    // Inventory with standard options
    mapTH = reader.tagInventory(true, (byte)0, (byte)1);
    for ( Map.Entry<String, FedmIsTagHandler> e : mapTH.entrySet() )
    {
        th = e.getValue();// TagHandler from HashMap

        // 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
        th = fedm.tagSelect(th, tagDriver);

        if(th instanceof FedmIsTagHandler_ISO15693)
        {
            FedmIsTagHandler_ISO15693 thIso = (FedmIsTagHandler_ISO15693)th;

            // read datablocks and write same data back
            back = thIso.readMultipleBlocksWithSecStatus(4, 4, res);
            back = thIso.writeMultipleBlocks(4, 4, 4, res.data);
        }
        else if(th instanceof FedmIsTagHandler_ISO14443_4_MIFARE_DESFire)
        {
            FedmIsTagHandler_ISO14443_4_MIFARE_DESFire thIso =
                (FedmIsTagHandler_ISO14443_4_MIFARE_DESFire)th;

            // read version information
            // use of the internal Interface IFlexSoftCrypto
            back = thIso.IFlexSoftCrypto.getVersion((byte)0, res);
        }
        else if(th instanceof FedmIsTagHandler_EPC_Class1_Gen2)
        {
            FedmIsTagHandler_EPC_Class1_Gen2 thGen2 = (FedmIsTagHandler_EPC_Class1_Gen2)th;
```

```
        // write new EPC to Transponder (withoutPassword)
        thGen2.writeEPC("0102030405060708090A0B0C", "");
    }
}
catch(java.lang.Exception e)
{
    ...
}
```

---

## 6. Basic properties of the class FedmlscFunctionUnit

---

The reader class methods can be roughly divided into five categories:

- a) Methods for initializing and finalizing
- b) Methods for data containers
- c) Methods for communication
- d) Methods for child list management

---

### 6.1. Initializing und Finalizing

---

---

#### 6.1.1. Initializing

---

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

Address of the Function Unit      The address of the function unit object must be set with the method `setPara(FEDM_ISC_FU_TMP_DAT_ADR, adr)`.

---

#### 6.1.2. Finalizing

---

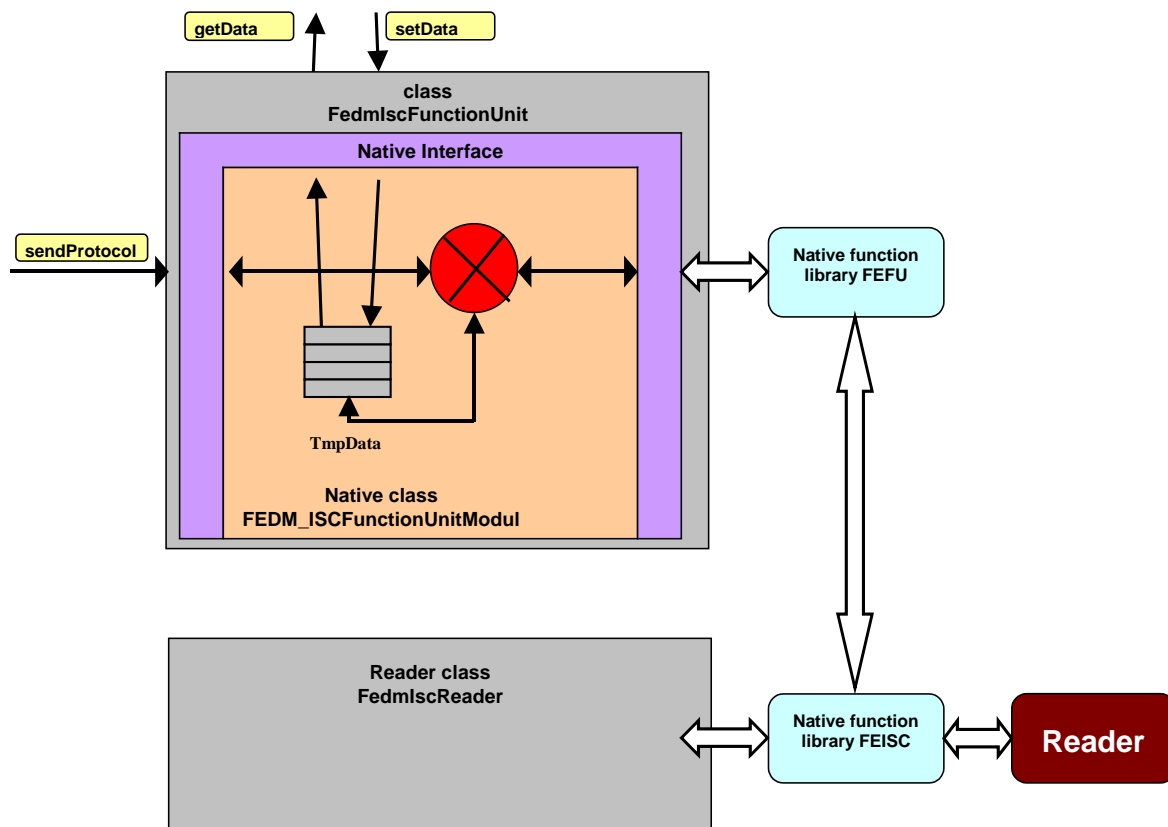
In Java the garbage collector assumes the task of removing no longer needed objects. This works wonderfully in pure Java applications. But objects that were created in native are not subject to the scrutiny of the garbage collector. Therefore the programmer must take over this work. In the class of this class library, this work is taken care of in one line: you invoke the reader class method *destroy* when you no longer need the reader object. If you omit this finalizing, you will get an exception no later than when the application is closed.

The *destroy* method destroys not the child function units objects, managed with the internal child table.

## 6.2. 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.



### 6.3.Examples for using the method `sendProtocol`

The method `sendProtocol` of the class is vitally important for the protocol transfer. For this reason an example is shown for each control byte, 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.

All access constants are contained in the interface **FedmlscFunctionUnitID** and should be studied thoroughly together with the explanation of protocol data contained in the system manual for the Function Unit.

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.

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

```
import de.feig.*;
```

The Function Unit object shall be defined as:

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

[Steuerbyte] Protokoll		Beispiel
ID ISC.DAT	[0xC0] Get Firmware Version	String firmware = new String; // buffer for firmware informations fu. <b>sendProtocol</b> ((byte)0xC0); firmware = fu. <b>getStringData</b> (FEDM_ISC_FU_TMP_SOFTVER);
	[0xC1] CPU Reset	fu. <b>sendProtocol</b> ((byte)0xC1);
	[0xC2] Set Capacities	fu. <b>setData</b> (FEDM_ISC_FU_TMP_DAT_ANT_VAL_C1, (byte)0xAB); // capacity 1 fu. <b>setData</b> (FEDM_ISC_FU_TMP_DAT_ANT_VAL_C2, (byte)0x9F); // capacity 2 fu. <b>sendProtocol</b> ((byte)0xC2);
	[0xC3] Get Antenna Values	String antValues = new String; // buffer for tuning values fu. <b>sendProtocol</b> ((byte)0xC3); antValues = fu. <b>getStringData</b> (FEDM_ISC_FU_TMP_DAT_ANT_VAL);
	[0xC4] Set Outputs	fu. <b>setData</b> (FEDM_ISC_FU_TMP_DAT_OUT, (byte)1); // switch output 1 fu. <b>sendProtocol</b> ((byte)0xC4);
	[0xC5] Re-Tuning	fu. <b>sendProtocol</b> ((byte)0xC5);
	[0xC6] Start Tuning	fu. <b>sendProtocol</b> ((byte)0xC6);
	[0xC8] Store Settings	fu. <b>sendProtocol</b> ((byte)0xC8);
	[0xC9] Detect	fu. <b>sendProtocol</b> ((byte)0xC9);



[Steuerbyte] Protokoll		Beispiel
	[0xCA] Set Address	<pre>byte newAdr = 2;           // new address fu.setData(FEDM_ISC_FU_TMP_DAT_NEW_ADR, newAdr); // new address for function unit fu.sendProtocol((byte)0xCA); // new address becomes valid fu.setData(FEDM_ISC_FU_TMP_DAT_ADR, newAdr); // set new address for communication</pre>
	[0xCB] Set Mode	<pre>fu.setData(FEDM_ISC_FU_TMP_DAT_MODE, (byte)1); // mode 1 fu.sendProtocol((byte)0xCB);</pre>
ID ISC.ANT.MUX	[0xDC] Detect	<pre>fu.sendProtocol(0xDC);</pre>
	[0xDD] Select Channel	<pre>fu.setData(FEDM_ISC_FU_TMP_MUX_OUT_CH 1, (byte)1); // set output 1 for input 1 fu.setData(FEDM_ISC_FU_TMP_MUX_OUT_CH 2, (byte)8); // set output 8 for input 2 fu.sendProtocol((byte)0xDD);</pre>
	[0xDE] CPU Reset	<pre>fu.sendProtocol((byte)0xDE);</pre>
	[0xDF] Get Firmware Version	<pre>String firmware = new String(); // buffer for firmware informations fu.sendProtocol((byte)0xDF); firmware = fu.getStringData(FEDM_ISC_FU_TMP_SOFTVER);</pre>
	[0xDC] Detect/Get Power	<pre>byte[] power; // 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); power = fu.getBytesArrayData(FEDM_ISC_FU_TMP_UMUX_POWER); UMuxStatus = fu.getBytesData(FEDM_ISC_FU_TMP_UMUX_LAST_STATE);</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((byte)0xDD); UMuxStatus = fu.getBytesData(FEDM_ISC_FU_TMP_UMUX_LAST_STATE);</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((byte)0xDE); UMuxStatus = fu.getBytesData(FEDM_ISC_FU_TMP_UMUX_LAST_STATE);</pre>
	[0xDF] Get Firmware Version	<pre>byte[] Firmware; // 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((byte)0xDF); Firmware = fu.getBytesArrayData(FEDM_ISC_FU_TMP_SOFTVER); UMuxStatus = fu.getBytesData(FEDM_ISC_FU_TMP_UMUX_LAST_STATE);</pre>
ID ISC.ANT.UMUX		

---

## 7. Error handling

---

---

### 7.1. 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	<a href="#">8.1.List of error codes</a>
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*.

---

### 7.2. Exceptions

---

Exceptions are generated in exceptional situations in the wrapper class for Java and during communication. The online documentation explains for each method whether and which exceptions are generated.

---

## 8. Appendix

---

---

### 8.1. List of error codes

---

All listed error codes are located in the Interface 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 constant	Value	Description
ERROR_UNKNOWN_READER_TYPE	-122	Unknown reader type
ERROR_UNSUPPORTED_READER_TYPE	-123	The Function doesn't support this reader type
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-Type
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 constant	Value	Description
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_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

---

## 8.2.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.PRHD102	all communication ports
ID ISC.MR200	all communication ports
ID ISC.LR200	
ID ISC.LR2000	all communication ports
ID ISC.LR2500-A	all communication ports
ID ISC.LR2500-B	all communication ports
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 CPR50.xx	all communication ports
ID CPR52.xx	all communication ports
ID MAX50.xx	all communication ports

---

### 8.3. 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

## 8.4. Revision history

---

### 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

### V4.06.16

- Discontinued support for Windows XP.

### V4.06.06

- TagHandler support for ISO 14443 Transponder with 10 byte UID
- Bugfix for EPC Class1 Gen2 Transponder with Extended PC
- Android Support for ARMv7-a, x86 and MIPS
- ISO 15693: [0x2C] Get Multiple Block Security Status with extended addressed mode: bugfix for received data above address 255

### V4.06.01

- Support for new Readers: : **ID ISC.PRH200, ID ISC.LRU1002, ID myAXXESS onTop**
- New transponder class: **FedmlscTagHandler\_ISO15693\_IDS\_SL13A**
- New method Convert\_EPC\_C1\_G2\_TagHandler in the Reader class **FedmlscReader**

### V4.05.04

- First Release with 64-Bit support for Windows and Linux
- New method in the Reader-class **FedmlscReader**: **readReaderDiagnostic**
- Bugfix for Linux: T=CL Support now activated
- Bugfix in Reader class **FedmlscReader** for **sendSAMCommand**

### V4.05.01

- First Release for USB with rooted Android



- **FedmlscTagHandler\_ISO14443\_4\_MIFARE\_DESFire\_FlexSoftCrypto:** Bugfix for data transfer in *readStandardData*, *writeStandardData*, *readRecords* and *writeRecords*
- **FedmlscTagHandler\_ISO14443\_4\_MIFARE\_DESFire:** Bugfix for false values in *ErrorSource* and *ErrorCode*

#### V4.05.00

- First Release for Android with ARMv5 Architecture
- Support for new Transponder: ISO 18000-3M3
- New TagHandler class: **FedmlscTagHandler\_ISO18000\_3M3**
- **FedmlscTagHandler\_ISO15693\_NXP\_ICODE\_SLI\_L:** new method *PasswordProtectAFI*
- New methods in TagHandler class **FedmlscTagHandler\_EPC\_Class1\_Gen2:**
  1. *getTagModelNumber*
  2. *getMaskDesignerID*
  3. *getMaskDesignerName*
  4. *isUidWithTid*
  5. *isExtendedPC\_W1*
  6. *getExtendedProtocolControlW1*
- Bugfix in *Kill* and *Lock* of **FedmlscTagHandler\_EPC\_Class1\_Gen2:** Calculation of password length fixed
- Class **FedmlscReader:**
  1. New method **GetDependentLibVersions**
  2. Modifications for method **StartAsyncTask**
    - a) While initializing the asynchronous Task for Reader's Notification-Mode, the Listener Port must be unused in the system. Otherwise, an exception with error code -4086 is thrown.
    - b) The Listener Port for Reader's Notification-Mode accepts only one connection at the same time. All additional connections will be rejected.
- Class **FedmBrmTableItem:** New element: *class1Gen2XPC\_W1* (Extended PC Word 1)
- Class **FedmlsoTableItem:** New element: *class1Gen2XPC\_W1* (Extended PC Word 1)
- New namespace **de.feig.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)
- Bugfix for Command [0x77] Get People Counter Values: false value in *counter2* corrected.
- Bugfix for Notification with wrong CRC
- Update of namespaces and access constants for reader configuration

V4.03.00

- Windows: removed dependency from FedmlscCoreVC60.DLL
- 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**:
  3. Support for Gate People Counter in Notification Mode
  4. New overloaded method *SetTableSize* to adjust additionally the buffers for Transponder data
  5. New, overloaded methods *ConnectCOMM* and *ConnectUSB* for secured data transmission.
- Class **FedmBrmTableItem**:
  1. Almost all data elements are now public for direct access
  2. Support for direction information in combination with Gate People Counter
  3. New table elements: IDDT, AFI and DSFID
- Class **FedmlsoTableItem**: Almost all data elements are now public for direct access
- Class **FelscReaderTime**: 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. <b>MifareAppID</b>	OperatingMode.xxMode.DataSource. <b>Mifare.Classic.AppID</b>
OperatingMode.xxMode.DataSource. <b>MifareKeyAddress</b>	OperatingMode.xxMode.DataSource. <b>Mifare.Classic.KeyAddress</b>
OperatingMode.xxMode.DataSource. <b>MifareKeyType</b>	OperatingMode.xxMode.DataSource. <b>Mifare.Classic.KeyType</b>

Note: xxMode stands for NotificationMode or ScanMode

V4.00.07

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

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 themethodFEDM\_ISCReaderModule::TagInventory
- TagHandler for EPC Class1 Gen2: new method ReadCompleteBank
- Bugfix in themethod FEDM\_ISCReaderModule::ReadCompleteConfiguration for ID ISC.LR2500-A and ID ISC.LRU3000

V4.00.00

- Update of namespaces and access constants for reader configuration
- Support for new Readers: **ID ISC.MRU102** and **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 firmwareversion 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 firmwareversionsless 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
< 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 **FedmlscReader**: *getTcpConnectionState*
- Support for [0x74] Input Event with Notification-Mode for the Reader **ID CPR50** and **ID MAX50**

- TagHandler-Class for EPC Class1 Gen2 with newmethods: *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.

### V3.03.00

- This version is not compatible with the previous version:
  1. FedmIscReader.getTableItem throws an exception of type FedmException
  2. FedmIscReader.getTable throws an exception of type FedmException
- 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**
- 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 FedmIscReader:
  1. Synchronous SAM-Command (SendSAMCommand)
  2. TagInventory
  3. TagSelect
  4. GetTagList
  5. GetTagHandler
  6. GetNonAddressedTagHandler
  7. GetSelectedTagHandler

### V3.02.06

- New reader configuration parameters in the package de.feig.ReaderConfig.

### V3.02.03

- Support for Bluetooth Reader under Linux (see [5.2. Administering the communications channels](#))

V3.02.02

- 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 `de.feig.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 `de.feig.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.
- Sample Projects for Eclipse 3.5 and Netbeans 6.8

V3.00.11

- New reader configuration parameters in the package `de.feig.ReaderConfig`.
- Linux: Adaptions in native libraries to latest Kernel versions for serial communication to prevent timeouts.
- Linux: Class **FedmCprAdu** for asynchronous ISO14443-4 T=CL protocol exchange with OBID® *classic-pro* Reader supports now Linux too.

V3.00.07

- - New methods in Reader class  
**FedmlscReader**: `transferReaderCfgToXmlFile`  
`transferXmlFileToReaderCfg`
- New reader configuration parameters in the package `de.feig.ReaderConfig`.
- Modifications in the Reader class **FedmlscReader**:
  - The methods `connectUSB` and `connectTCP` execute internally a `readReaderInfo` to collect important Reader properties.
  - The method `connectCOMM` open a serial port and can optional, but recommended, execute internally a `findBaudrate` and after this, if the Reader is detected successfully, a `readReaderInfo` to collect important Reader properties.
  - The method `addCommand` is renamed in `addCommandToQueue`

- The method `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 addresses one bit. Now they addresses three bits. Thus, the OUT-TYPE 'Relay' must be set to 0x04 instead of 0x01 ([5.7. Examples for using the method `sendProtocol`](#)). This is applied to all reader types which supports the command [0x72] Set Output.
- Bugfix: Semaphore blocker in **FedmlscReader**.connectXXX methods solved.
- Bugfix: **FedmlscReader**.set/getTableItem methods evaluate different length of UID (serial number).

#### V3.00.04

- New reader configuration parameters in the package `de.feig.ReaderConfig`.
- Linux: the native libraries are linked against Libc V6 and Libstdc++ V6

#### 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 class **FedmlscReaderInfo** collecting important information from the connected reader.
- Collecting of all access constants for reader configuration in them package `de.feig.ReaderConfig` improves the clearness. Thus, the interfaces `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 interface `FedmlscReaderID` are removed.
- New overloaded methods `getConfigParaAsXXX/setConfigPara` in class **FedmlscReader** for modifying reader configuration parameters in the package `de.feig.ReaderConfig`.
- Writing of reader configuration is only possible for previous read configuration blocks except, if the reader configuration is load by a XML file.
- New high-level methods in **FedmlscReaderModule**
  - `ApplyConfiguration`
  - `ReadCompleteConfiguration`
  - `WriteCompleteConfiguration`
  - `ResetCompleteConfiguration`
  - `ReadReaderInfo`

- Removed methods in class **FedmlscReader**:
  - void **setByteContainer** (int arrayID, byte[] array)
  - byte[] **getByteContainer** (int arrayID)
  - int **setByteArrayData** (int address, byte[] data, int memID)
  - byte[] **getByteArrayData** (int address, int length, int memID)
- New class **FedmCprAdu** for asynchronous ISO14443-4 T=CL protocol exchange with OBID®*classic-pro* Reader (only for Windows).
- New class **FedmCprCommandQueue** for OBID®*classic-pro* Reader for asynchronous execution of [0xBC] Command Queue.

#### V2.05.07

- Support for USB reader
- Support for a new UHF reader command: [0x6B] Centralized RF Synchronization
- Linux: all native libraries are compiled with the GNU Compiler Collection 3.3.3.

#### V2.05.01

- Modified licence agreement
- Support for the UHF-Reader ID ISC.LRU2000.
- Extensions for the UHF-Reader ID ISC.LRU1000 concerning the configuration.
- New methods in the Reader class **FedmlscReader** for supporting asynchronous tasks.
- New interface **FedmTaskListener**
- New property class **FedmTaskOption**

#### V2.04.00

- 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	

Constant	Comment
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

- The file **feusb.properties** is obsolete
- 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 **FedmIscReader**:
  1. sendProtocol (byte cmdByte, String requestData)
  2. sendTransparent (String requestProtocol, boolean calcCrc)
- 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



Old Constant	New Constant
FEDM_ISC_TMP_ADV_BRM_TRDATA1_...	FEDM_ISC_TMP_ADV_BRM_TRDATA_...

### V2.03.00

- Support for the new Reader ID ISC.MR200, ID ISC.MR/PR101 and ID CPR.M02-U
- Support for external Function Units with the class **FedmlscFunctionUnit**.
- The reader class **FedmlscReader** checks the version number of the native library OBIDISC4J.DLL/libobidisc4j.so. A version conflict throws an exception.
- The reader class **FedmlscReader** is no longer derived from the interfaces FedmlscReaderID, FedmlscReaderID\_LR200 and FedmlscReaderID\_LRU1000.
- New method in the reader class **FedmlscReader**: getNativeLibVersion
- Modifications in the table class **FedmBrmTableItem**: the datatype of the member blockCount is changed from byte into int. This causes adaptations in the get/set methods of this class. If you use up to now the method getByteData to query the value of the member blockCount (with constant DATA\_DBN), then you must change the method call into getIntegerData.
- New member in the table class **FedmBrmTableItem**: member blockSize (access with constant DATA\_BLOCK\_SIZE) for supporting the Advanced Buffered Read Mode of the UHF-Reader ID ISC.LRU1000.
- New constants in the interface **FedmlscReaderConst** for UHF-Transponders.
- New constants for the UHF-Reader ID ISC.LRU1000 for the configuration CFG16 in interface **FedmlscReaderID\_LRU1000**.
- New general constants:

Identifier	Comment
FEDM_ISC_TMP_DIAG_DATA	This identifier is valid for all reader diagnostic modes and substitutes the below listed removed constants for the modes 0x01, 0x02, 0x03.
FEDM_ISC_TMP_B3_...	Identifier for [0xB3] commands

- Modified general constants:

Old Identifier	New Identifier
FEDM_ISCLR_TMP_DIAG_MODE	FEDM_ISC_TMP_DIAG_MODE

- Removed general constants:

Identifier	Comment
FEDM_ISCLR_TMP_DIAG_0x01_DATA	Substituted through FEDM_ISC_TMP_DIAG_DATA
FEDM_ISCLR_TMP_DIAG_0x02_DATA	Substituted through FEDM_ISC_TMP_DIAG_DATA
FEDM_ISCLR_TMP_DIAG_0x03_DATA	Substituted through FEDM_ISC_TMP_DIAG_DATA

Identifier	Comment
FEDM_ISC_TMP_EPC_DESTROY_LEN	Removed, because of internal calculation of the length of EPC/UID based on the destroy mode and the header of the EPC.

### V2.02.00

- Support for new reader ID ISC.MR200
- Protocols [0x18] Destroy supports new transponder type I-CODE UID.
- Remove of all access constants for data container RAMDATA\_MEM. This reduces the number of constants dramatically. Alternatively, one can use the method toRAM of class FeMethods to modify the access constant for data container EEDATA\_MEM.

### V2.01.00

- Support for new reader ID ISC.LRU1000
- Support of new protocols [0x18] Destroy and [0x22] Read Buffer
- Support for advanced protocol frames (> 255 bytes)
- New helper class FeMethod with methods for access constants
- Rename of obid.dll (libobid.so) in obidisc4j.dll (libobidisc4j.so)
- Move of access constants concerning the configuration parameters in separate file for the readers ID ISC.LR200 and ID ISC.LRU1000

### V2.00.05

Error correction in the class FeUsb

### V2.00.04

Add of new item antennaNumber to the class FedmBrmTableItem

### V2.00.03

This is the first release version.