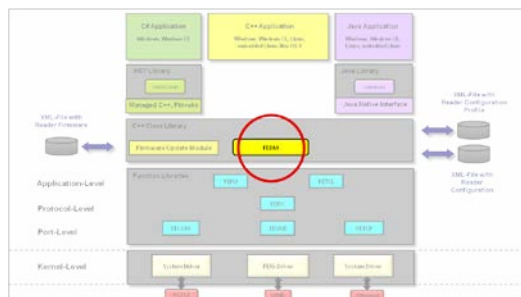


# C++ Class Library ID FEDM

Version 4.05.00

## Part A

### Software-Support for OBID® Reader Families



Operating System	Target		Notes
	32-Bit	64-Bit	
Windows XP	X	(X)	with 64-Bit OS: only with 32-Bit Runtime Environment
Windows Vista / 7	X	X	
Windows CE	X	-	
Linux	X	(X)	with 64-Bit OS: only with 32-Bit Runtime Environment
Apple Max OS X	-	X	OS X V10.7.3 or higher Architecture x86_64

## Note

© Copyright 2001-2013 by FEIG ELECTRONIC GmbH  
Lange Straße 4  
D-35781 Weilburg-Waldhausen  
eMail: [obid-support@feig.de](mailto:obid-support@feig.de)

This manual supercedes all previous editions.  
The information contained in this manual is subject to change without notice.

**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.**

The information contained in this manual has been gathered with all due care and to the best of our knowledge. FEIG ELECTRONIC GmbH assumes no liability for the accuracy and completeness of the data in this manual. In particular, FEIG ELECTRONIC GmbH cannot be held liable for consequential damages resulting from inaccurate or incomplete information. Since even with our best efforts this document may still contain mistakes, please contact us should you find any errors.

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

The installation instructions given in this manual are based on advantageous boundary conditions. FEIG ELECTRONIC GmbH does not give any guarantee promise for perfect function of an OBID®-system in cross surroundings.

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

Linux® is a registered trademark of Linus Torvalds.

Apple, Mac, Mac OS, OS X, Cocoa and Xcode are trademarks of Apple Inc., registered in the U.S. and other countries.

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

I-CODE® and Mifare® are registered Trademarks of Philips Electronics N.V.

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

## Licensing agreement for use of the software

This is an agreement between you and FEIG ELECTRONIC GmbH (hereafter "FEIG") for use of the ID FEDM C++ Class Library and the present documentation, hereafter called licensing material. By installing and using the software 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. This licensing material remains the property of FEIG ELECTRONIC GmbH and is protected by international copyright.

### §1 Object and scope of the agreement

1. FEIG grants you the right to install the licensing material provided and to use it under the following conditions.
2. You may install all components of the licensing material on a hard disk or other storage medium. The installation and use may also be done on a network fileserver. You may create backup copies of the licensing material.
3. FEIG grants you the right to use the documented program library for developing your own application programs or program libraries, and you may sell runtime files without licensing fees under the stipulation that these application programs or program libraries are used to control devices and/or systems which are developed and/or sold by FEIG.

### §2. Protection of the licensing material

1. The licensing material is the intellectual property of FEIG and its suppliers. It is protected in accordance with copyright, international agreements and relevant national statutes where it is used. The structure, organization and code of the software are a valuable business secret and confidential information of FEIG and its suppliers.
2. You agree not to change, modify, translate, reverse develop, decompile, disassemble the program library or the documentation or in any way attempt to discover the source code of this software.
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 transmission of licensing material in part or in full is prohibited unless there is an explicit agreement to the contrary between you and FEIG. Application programs or program libraries which are created and sold in accordance with §1 Par. 3 of this Agreement are excepted.

### §3 Warranty and liability limitations

1. You agree with FEIG that it is not possible to develop EDP programs such that they are error-free for all application conditions. FEIG explicitly makes you aware that the installation of a new program can affect already existing software, including such software that does not run at the same time as the new software. FEIG assumes no liability for direct or indirect damages, for consequential damages or special damages, including lost profits or lost savings. If you want to ensure that no already installed program will be affected, you should not install the present software.
2. FEIG explicitly notes that this software makes it possible for irreversible settings and adaptations to be made on devices which could destroy these devices or render them unusable. FEIG assumes no liability for such actions, regardless of whether they are carried out intentionally or unintentionally.
3. FEIG provides the software „as is“ and without any warranty. FEIG cannot guarantee the performance or the results you obtain from using the software. 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 Concluding provisions**

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 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.
4. This agreement is subject to the laws of the Federal Republic of Germany. Place of jurisdiction is Frankfurt a. M.

## Contents:

<b>1. Introduction</b>	<b>8</b>
1.1. Overview of all OBID components	10
1.2. Supported 32- and 64-Bit Operating Systems	11
1.3. Supported Development Environments	11
<b>2. Changes from the previous version</b>	<b>12</b>
<b>3. Installation</b>	<b>13</b>
<b>4. Overview</b>	<b>14</b>
4.1. Class tree	14
4.2. Class structure diagram	14
4.3. Component diagram	16
4.4. Thread security	17
<b>5. Basic properties of the class library</b>	<b>18</b>
5.1. Internal structure	18
5.2. Data containers	19
5.2.1. Data exchange	20
5.3. Access constants for temporary protocol data	22
5.4. Namespaces for reader configuration parameters	23
5.5. Tables	24
5.6. Protocol traffic	24
5.7. Initialization methods	25
5.8. Serializing	25
5.9. Error handling	26
5.10. Language support	26
<b>6. Class description</b>	<b>27</b>
6.1. FEDM_Base	27
6.1.1. Methods (public)	27
6.2. FEDM_DataBase	28
6.2.1. Attribute (public)	28
6.2.2. Methods (public)	29
6.2.3. Abstract methods (public)	29
6.3. FEDM_XMLBase	31

6.3.1. Methods (public) .....	31
<b>6.4. FEDM_XMLReaderCfgDataModul .....</b>	<b>32</b>
6.4.1. Methods (public) .....	32
<b>7. Global functions .....</b>	<b>33</b>
7.1. FEDM_Functions .....	33
<b>8. Appendix .....</b>	<b>36</b>
8.1. Class Tree .....	36
8.2. List of error codes.....	37
8.3. List of reader families .....	40
8.4. List of protocol type constants .....	40
8.5. List of language constants .....	40
8.6. List of memory type constants.....	40
8.7. Macros .....	42
8.8. Revision history .....	43

## Notes concerning the documentation for this library

This manual describes a software library which is also available as annotated source code. For this reason we have limited the documentation to what is absolutely necessary for understanding the functionality and use of the classes. It is assumed that the user of this library reads the source code and becomes familiar with the details using this document, the header files and the included comments.

To understand the internal program sequences you will also have to refer to the system manuals for whichever OBID® readers and OBID® function libraries you are using.

FEIG ELECTRONIC GmbH does not repeat the same information about OBID® readers in different manuals or use cross-references to certain pages in a different document. This is necessary due to the constant updating of manuals, and it prevents confusion caused by information in out-of-date documents. The user of this library is therefore well advised to check regularly that he has the most current manuals. If not, these can of course be obtained whenever needed from FEIG ELECTRONIC GmbH.

### **Important notes:**

**You are only permitted to use this library if you have first agreed to the license conditions on the back of this page.**

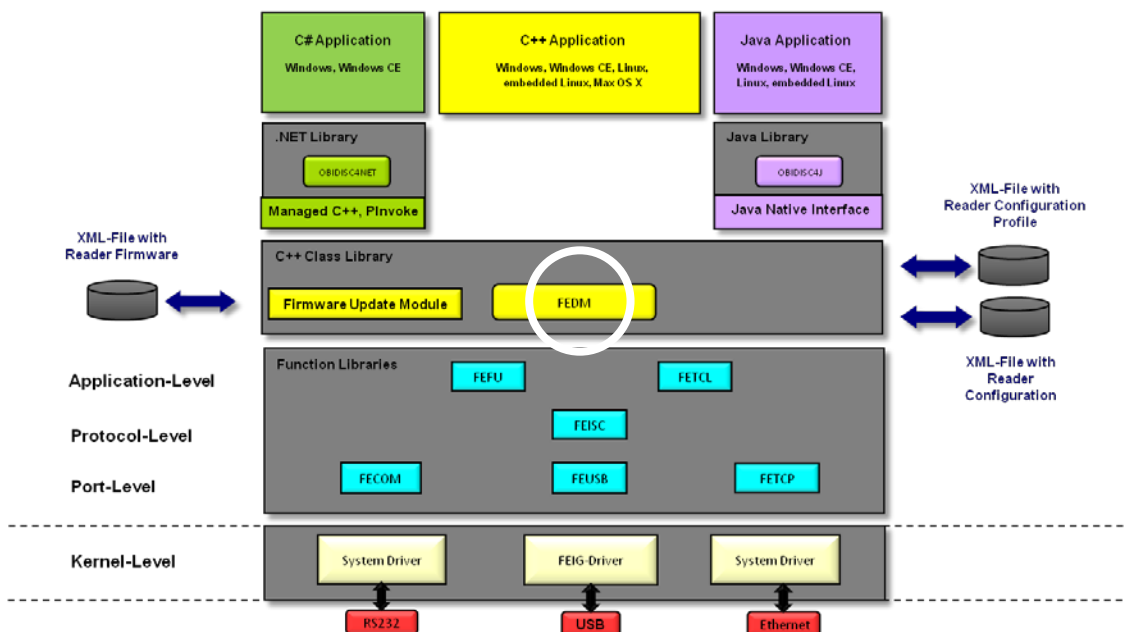
**Anyone is free to modify source code. Therefore you should work only with libraries you have received directly from FEIG ELECTRONIC GmbH. In any case further transmission of the source code is prohibited.**

## 1. Introduction

With its C++ Class Library ID FEDM FEIG ELECTRONIC GmbH provides you with another component for simplifying development of user programs for OBID-RFID readers.

The C++ Class Library ID FEDM supports all OBID® reader families and can be considered an additional protocol layer above the OBID® function libraries.

The library FEDM is for C++ highest level of a hierarchical structured, multi-tier FEIG library stack. The following picture shows the multi-tier library stack.



The C++ Class Library ID FEDM is the introduction of an organizational principle for all OBID® reader families which allows you to create similar program structures for all OBID® readers regardless of the reader you are using. This organizational principle, which has been implemented in the form of similar function interfaces in the OBID® function libraries as well, is here expanded to data containers and control of the data stream and protocol traffic.

In spite of the uniform organizational principle, the view of the storable reader and transponder data is still at a very low level. This means that as a programmer you are confronted with reader parameters in bits and bytes and are offered transponder data only in the form of unorganized data quantities. The advantage of this is that you have access to everything, but on the other hand you have to carry out multiple operations in sequence if you want for example to write just a small amount of data to a transponder. Additional simplification with respect to abstraction of data streams and actions remains reserved for a higher-order module layer.

The C++ class library ID FEDM offers a simple way of serializing data for the reader configuration. This makes it possible to store a complete reader configuration in an XML file, load it again later and transfer it to the reader.



The documentation for the C++ Class Library is in two parts: This document described only the base classes and common features of the specialized reader classes. A detailed description of the reader classes is contained in separate manuals.

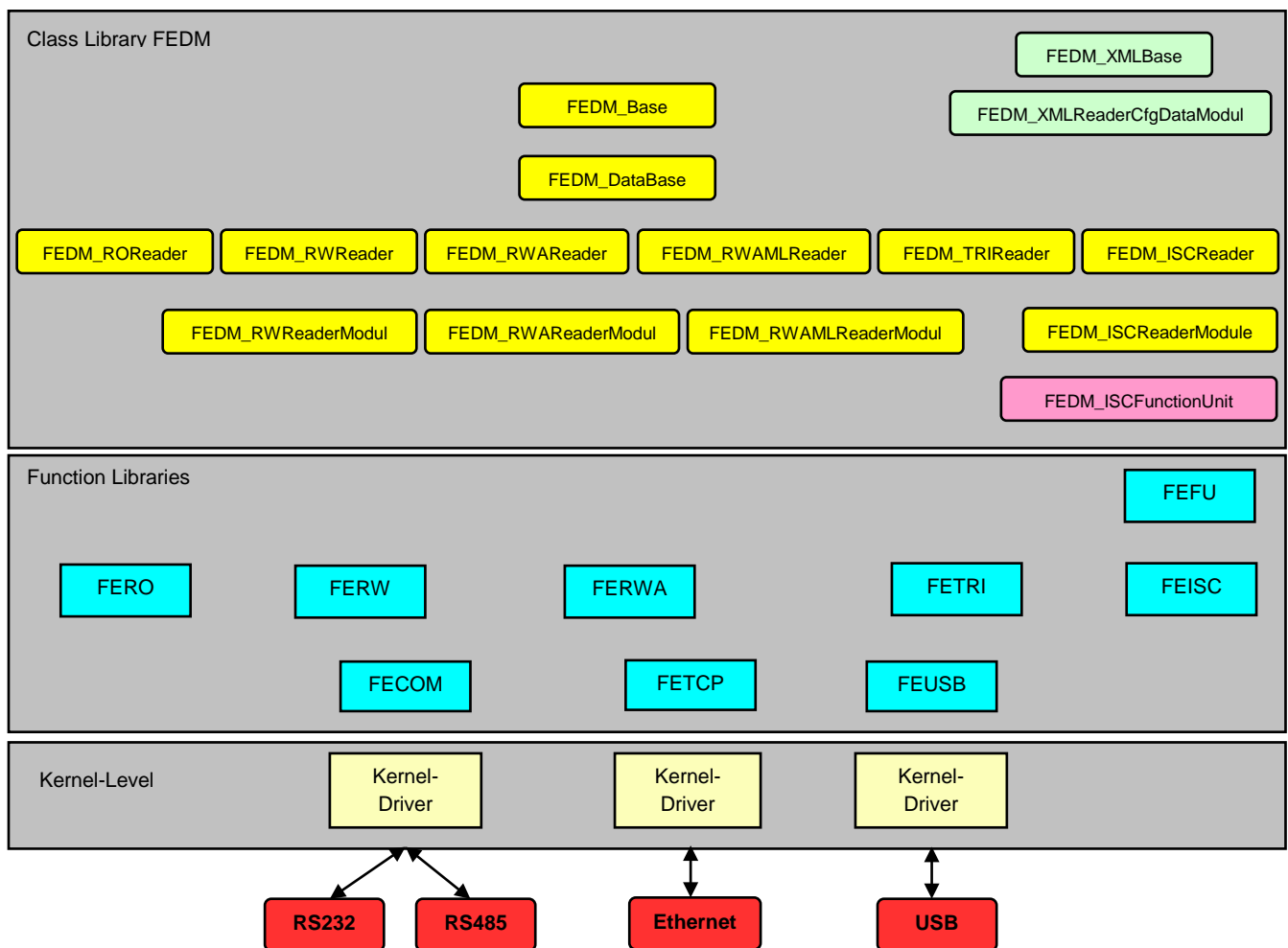
**Important note:**

The ID FEDM class library undergoes a continuous adaptation process. We will make every effort to retain the documented status. Bear in mind that changes are still possible however.

## 1.1. Overview of all OBID components

The C++ Class Library ID FEDM builds upon other OBID® components. The overview shows which other components are necessary for the respective reader family and intended interface. The function libraries are all in the form of dynamic linked libraries (DLL or LIB).

Information on the OBID® function libraries can be found in the respective manuals. FEIG ELECTRONIC will be happy to provide you with these documents on request in case you are missing one.



The layering of libraries, which is evident in the overview, reflects the degree of specialization. The lowest (physical) level with the kernel drivers takes over hardware-proximate protocol transfer. The overlying protocol transfer layer with **FECOM**, **FETCP** and **FEUSB.DLL** provides an application with the first function interface for the physical interfaces. An additional overlying layer contains the OBID® reader-specific protocol layer (**FERO**, **FERW**, **FERWA**, **FETRI**, **FEISC**). This layer already allows simplified communication with OBID® readers.

In more complex applications you must construct organization forms for the data transferred via protocols. This is the job of the ID FEDM C++ Class Library. It in turn is located one layer above the OBID® reader-specific protocol layer. It must now provide organization structures for all OBID®

reader families. This objective is carried out using the base classes, the specialized reader classes and the use of object-oriented methods such as overlaid methods, abstract base class (FEDM\_DataBase) and use of the Standard Template Library (STL).

The ID FEDM C++ Class Library does not have to be integrated into an application in its complete form. Depending on the OBID® reader family you may use only the specialized reader class and base classes along with the global functions and constants. The lower-level function libraries for your reader class are however mandatory for the functionality of the reader class within the ID FEDM C++ Class Library.

## 1.2. Supported 32- and 64-Bit Operating Systems

Operating System	Target		Notes
	32-Bit	64-Bit	
Windows XP	X	(X)	with 64-Bit OS: only with 32-Bit Runtime Environment
Windows Vista / 7	X	X	
Windows CE	X	-	
Linux	X	(X)	with 64-Bit OS: only with 32-Bit Runtime Environment
Apple Max OS X	-	X	OS X V10.7.3 or higher Architecture x86_64

## 1.3. Supported Development Environments

Operating System	Development Tool	Supported
Windows XP / Vista / 7	Visual Studio 6	on request
	Visual Studio 2005 / 2008 / 2010	yes, Professional Version or higher required
	Borland C++ Builder	on request
	Embarcadero C++ Builder	on request
Windows CE	eMbedded Visual C++ 4	no
	Visual Studio 2005 / 2008	yes, Professional Version or higher required
Linux	GCC	yes, for 32-Bit projects
Mac OS X	GCC	yes, for projects with x86_64 architecture
	Xcode ≥ V4.3.2	yes, for projects with x86_64 architecture

## 2. Changes from the previous version

---

- New global helper functions:

**FEDM\_ConvBcdCharToHexUChar** and **FEDM\_ConvHexUCharToBcdChar**

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

---

### 3. Installation

---

The installation is described in detail in part B (H10202-xe-ID-B) of class library documentation.

---

## 4. Overview

---

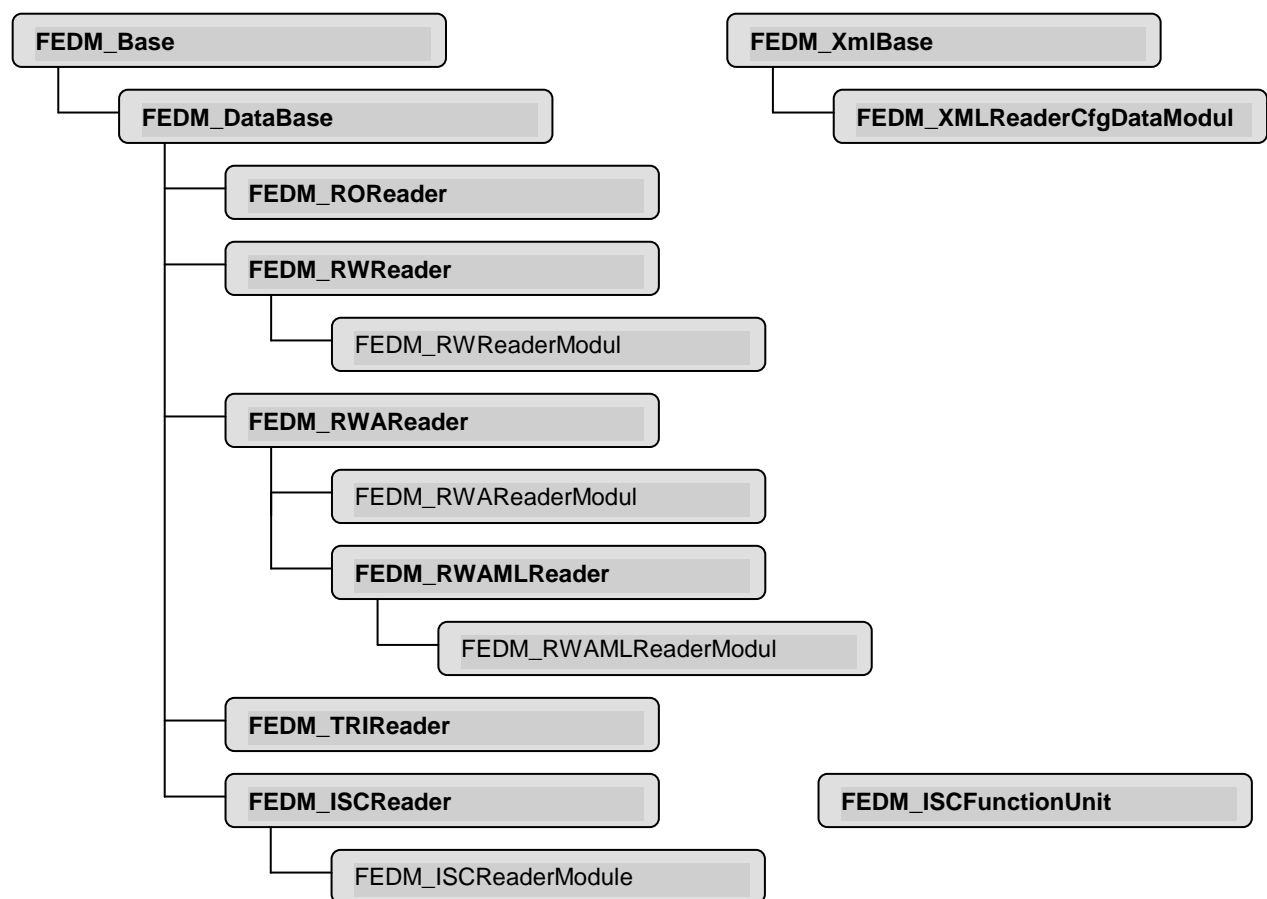
Depending on the user's requirements he will need a base class upon which to build his special reader class, or he uses the existing reader class as a component with a defined interface – something like a black box object. A diagram is shown below for each of these two application scenarios.

---

### 4.1. Class tree

---

Class tree for a first overview. The classes FEDM\_RWReaderModul, FEDM\_RWARReaderModul and FEDM\_RWAMLReaderModul are currently still under construction.



---

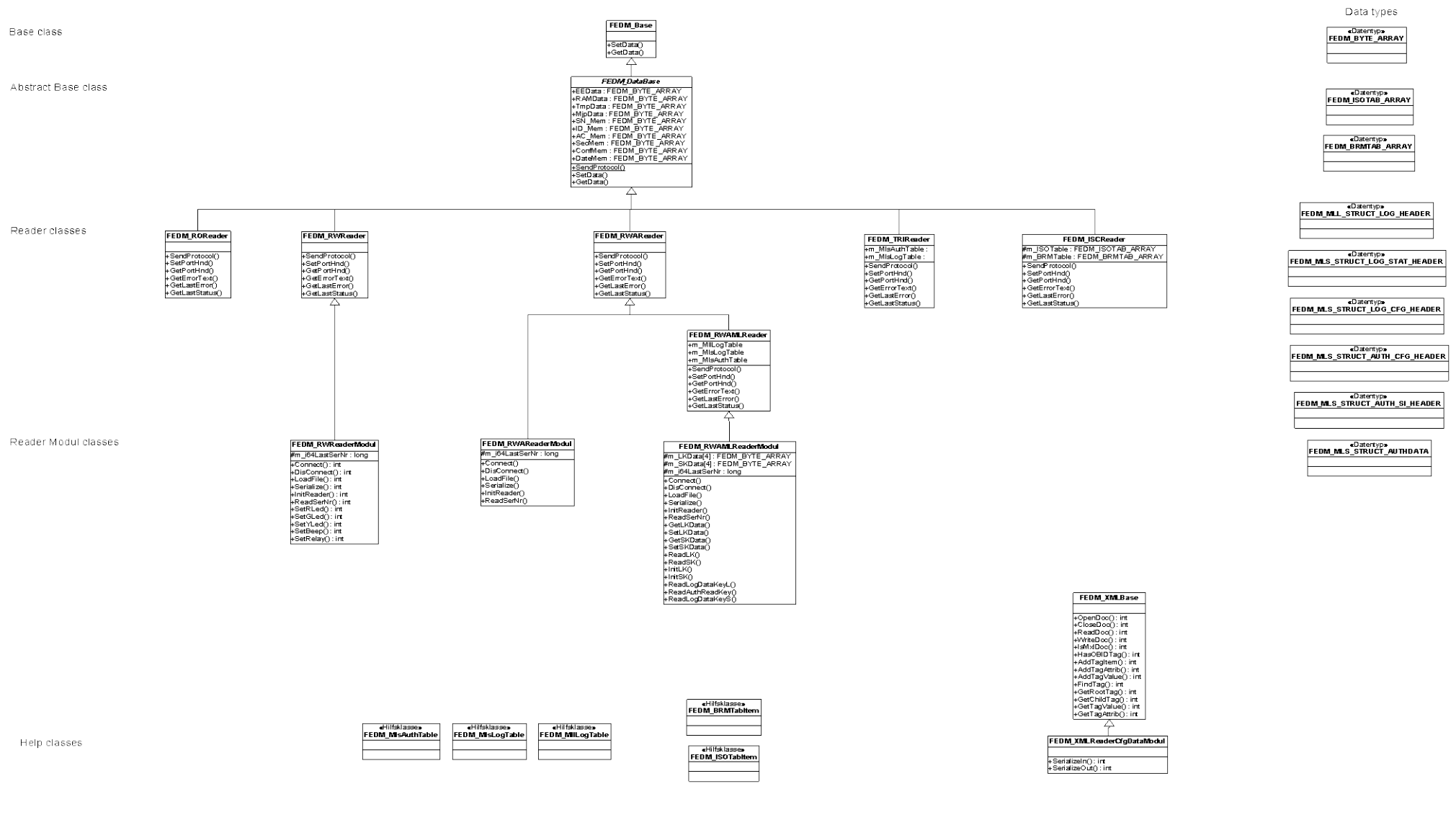
### 4.2. Class structure diagram

---

The structure diagram shows the static structure of the C++ Class Library, class tree, FEDM-specific data types and help classes.

The classes in the diagram contain only a few of the important attributes and methods.

## Static class structure ID FEDM

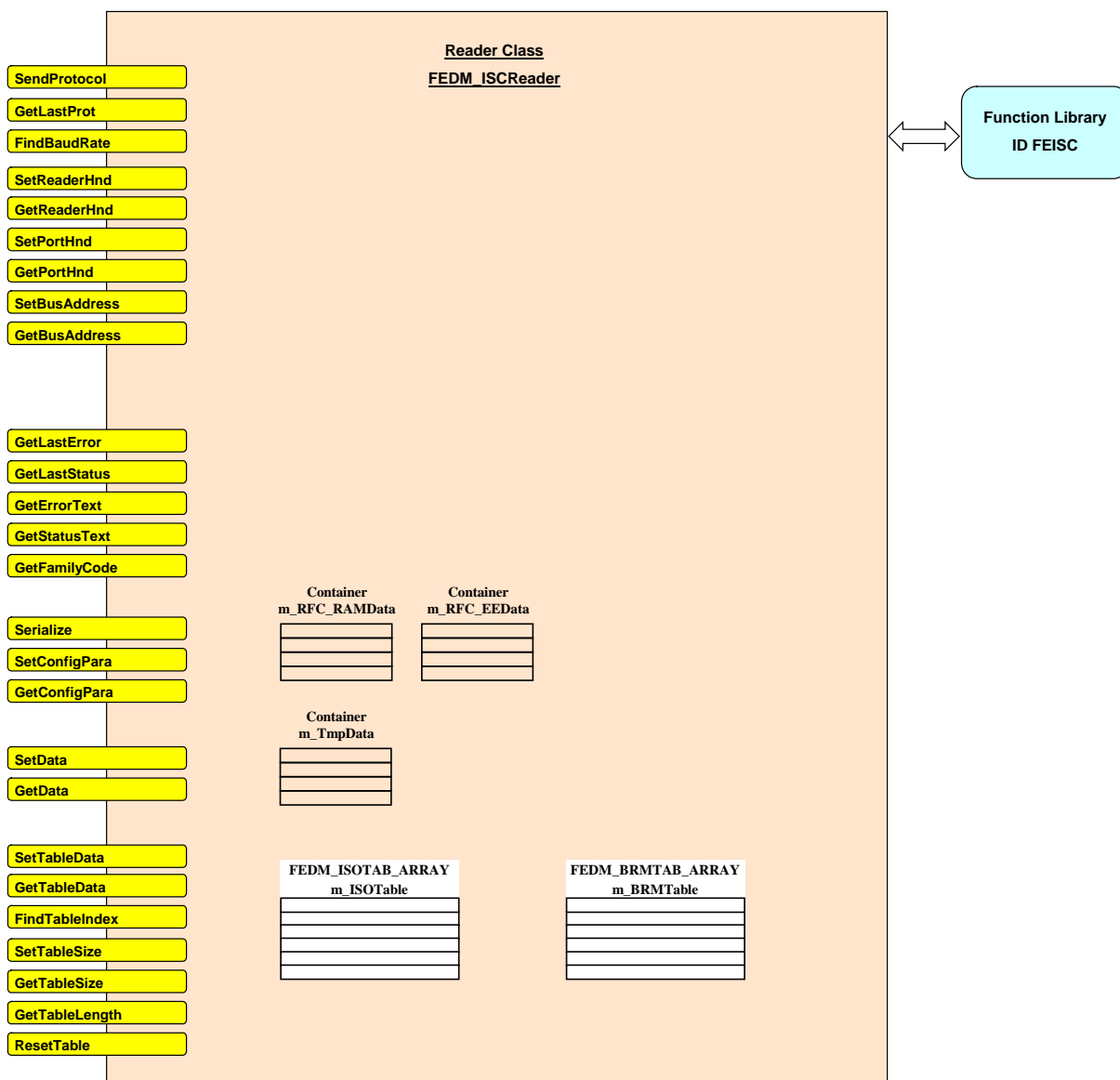


### 4.3. Component diagram

The component diagram shows another view of the ID FEDM C++ Class Library.

Only the most important methods are shown. Attributes have been omitted. For a more exact and complete component diagram, see the documentation for the respective reader class.

Most of the methods represent an interface for all reader classes which is uniform and independent of type. This makes it possible to run various reader families in an application using the same algorithms.





#### 4.4. Thread security

---

In principle, all FEIG libraries are not fully thread safe. But respecting some guidance, a practical thread security can be realized allowing parallel execution of communication tasks. One should keep in mind, that all OBID® RFID-Reader works synchronously and can perform commands only in succession.

On the level of the transport layer (FECOM, FEUSB, FETCP) the communication with each port must be synchronized in the application, as the Reader works synchronously. Using multiple ports and so multiple Readers from different threads simultaneously is possible, as the internal port objects acts independently from each other. But it is not possible to communicate independently from different threads with different Readers over one serial port of type RS485 or RS422. Yet another limitation concerns the Scan function of FEUSB library. The scan over the complete USB cannot be thread-safe, as a global kernel action is performed. To prevent mutual interactions, the opening and closing of serial and USB connections must be serialized on application side.

On the level of the protocol layer (FEISC), parallelism can be realized only when each Reader object represents exactly one physical Reader and is bound with an individual communication port. This is not true for the four specialized functions FEISC\_BuildxxProtocol and FEISC\_SplitxxProtocol, which use an internal global buffer for protocol data.

The library FEFU has no precautions for thread-safeness implemented. Thus, only one thread can call FEFU functions at the same time. Thread-safeness must be implemented on application side.

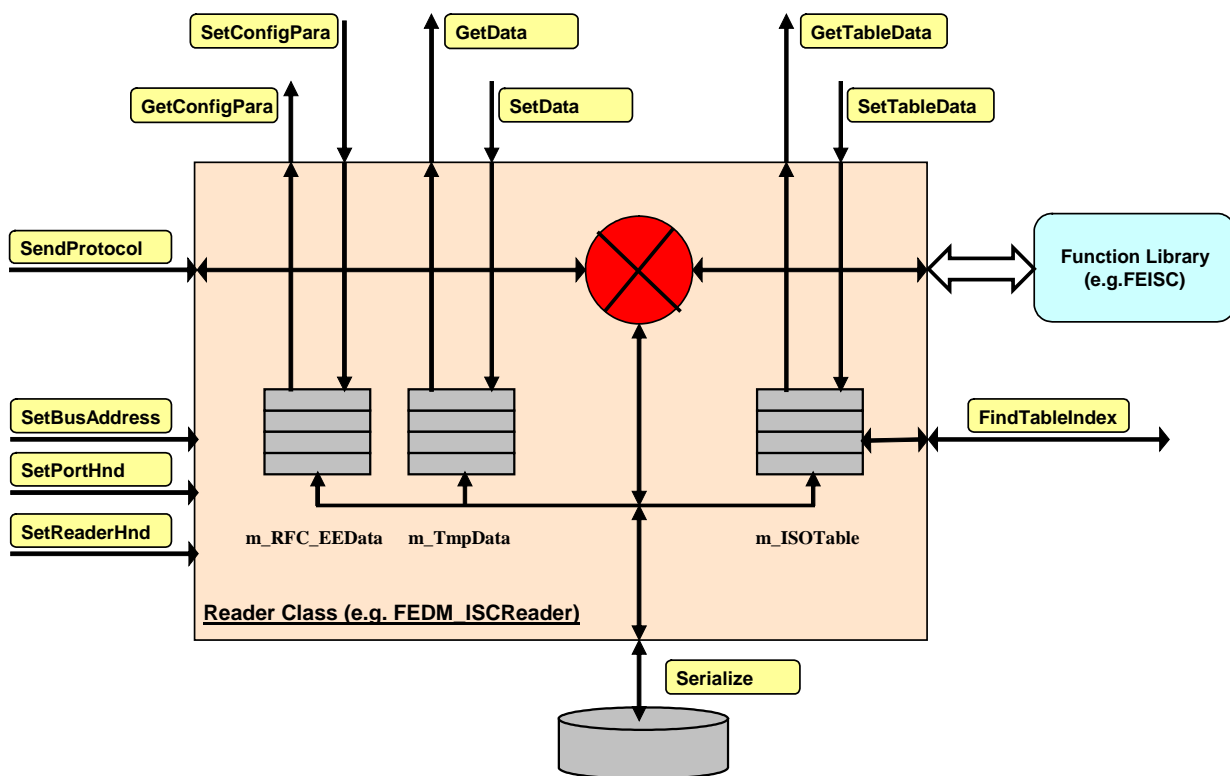
The library FETCL for ISO 14443-4 compliant Transponders is thread-safe, only when each Transponder object is connected with a different Reader object and only one APDU is exchanged with each Reader at the same time. Even if the function FETCL\_Apdu can be called asynchronously, this means not, that multiple calls of FETCL\_Apdu to the same Transponder object are allowed. APDUs are not stored in a stack.

On the level of the C++ class library FEDM, parallelism can be realized when with each Reader object only one method call is performed. Thread-safeness for each Reader object must be implemented on application side. Parallelism with non-synchronized opening and closing of serial and USB ports (ConnectCOMM, ConnectUSB) must be avoided.

## 5. Basic properties of the class library

### 5.1. Internal structure

The function of a reader class – to which you can ultimately reduce the view when using the class library – can be clearly seen in the following illustration: In the vertical axis are the data streams which are moved using the (overlaid) methods GetData respectively GetConfigPara and SetData respectively SetConfigPara, as well as GetTableData<sup>1</sup> and SetTableData. You can also use the Serialize method to move data between a reader class object and a file.



The horizontal axis shows the control flow that is generated by the SendProtocol method, the only communication method. It independently retrieves all the necessary data from the integrated data containers before transmitting the send protocol and stores the received protocol data there as well. This means the application must write all the data necessary for this protocol to the corresponding data containers in the correct locations before invoking the SendProtocol. Likewise the receive data are stored at particular locations in corresponding data containers.

The key to the protocol data are so-called access constants for temporary protocol data (e.g. FEDM\_ISC\_TMP\_READER\_INFO\_MODE) and namespaces for reader configuration parameters (e.g.

<sup>1</sup> Not all reader classes have an implementation of GetTableData and SetTableData, i.e. those without an integrated table.

`ReaderConfig::OperatingMode::Mode`). Anywhere from a few dozen to a hundred constants and namespaces can be defined for each reader class. The structure is the same for all reader classes and is especially significant. This is described in detail in section [5.3. Access constants for temporary protocol data](#) and [5.4. Namespaces for reader configuration parameters](#). Since the access constants are of essential important for the function of the protocol transfer, they are described in detail in the documentation for the reader classes. The definition of each reader configuration parameter in a namespace is documented in the system manual of the reader.

---

## 5.2. Data containers

---

Data containers have the job of structurally administrating all the parameters and transponder data. Internally all data containers are organized as byte arrays in Motorola format (Big Endian). This format corresponds to each OBID reader. Conversion into the Intel format required for Intel-based PC's (Little Endian) is handled by the overlaid access methods.

The byte arrays are organized in blocks of 32, 16 or 4 bytes each. This organization also corresponds to the reader and transponder.

A total of 11 data containers are integrated in the abstract base class `FEDM_DataBase`. They all have the size 0. The size of the required containers is determined by the derived reader class. Unused data containers have a size of 0.

Data container	Description
<code>m_RFC_EEData</code>	for reader configuration parameters
<code>m_RFC_RAMData</code>	for temporary reader configuration parameters
<code>m_ACC_EEData</code>	for reader configuration parameters
<code>m_ACC_RAMData</code>	for temporary reader configuration parameters
<code>m_TmpData</code>	for general temporary protocol data

Data container especially for classic reader family (RW/RWA)

Data container	Description
<code>m_MjpData</code>	for temporary protocol data of a multijob-Poll
<code>m_SN_Mem</code>	for transponder serial numbers
<code>m_ID_Mem</code>	for transponder ID numbers
<code>m_AC_Mem</code>	for transponder account data
<code>m_PubMem</code>	for transponder public data blocks
<code>m_SecMem</code>	for transponder secret data blocks
<code>m_ConfMem</code>	for transponder configuration data
<code>m_DateMem</code>	for transponder date data

---

### 5.2.1. Data exchange

---

Access to the temporary protocol data is accomplished primarily using the overlaid methods SetData and GetData. Each method invoke allows exactly one parameter to be read or written, which is identified by an access constant ([5.3. Access constants for temporary protocol data](#)).

The data exchange with reader configuration parameters is accomplished primarily using the overlaid methods SetConfigPara and GetConfigPara.

Alternately you can directly access the bytes in a data container, since they are stored in the class FEDM\_DataBase public. This method should only be used however for accessing by the byte.

The following chapters illustrate the use of GetData and SetData. The use of GetConfigPara and SetConfigPara is analogous with the difference that the access constant is replaced by the string of the namespace.

---

#### 5.2.1.1. Constant data

---

```
int iErr = SetData(FEDM_ISC_TMP_READ_CFG_MODE, false);           // bool
int iErr = SetData(FEDM_ISC_TMP_READ_CFG_MODE, FALSE);          // BOOL
int iErr = SetData(FEDM_ISC_TMP_READER_INFO_MODE, (UCHAR)0x01);  // unsigned char
int iErr = SetData(FEDM_ISC_TMP_READER_INFO_MODE, (UINT)0x001);  // unsigned int
int iErr = SetData(FEDM_ISC_TMP_READER_INFO_MODE, (CString)"0001"); // CString bzw. AnsiString
int iErr = SetData(FEDM_ISC_TMP_READER_INFO_MODE, (string)"0001"); // STL-string
```

---

#### 5.2.1.2. Data type bool

---

```
bool bData = false;
int iErr = GetData(FEDM_ISC_TMP_INP_STATE_IN1, &bData);
int iErr = SetData(FEDM_ISC_TMP_READ_CFG_MODE, bData);
```

---

#### 5.2.1.3. Data type BOOL

---

```
BOOL bData = FALSE;
int iErr = GetData(FEDM_ISC_TMP_INP_STATE_IN1, &bData);
int iErr = SetData(FEDM_ISC_TMP_READ_CFG_MODE, bData);
```

---

#### 5.2.1.4. Data type unsigned char (UCHAR)

---

```
UCHAR ucData = 0x01;
int iErr = GetData(FEDM_ISC_TMP_INP_STATE, &ucData);
int iErr = SetData(FEDM_ISC_TMP_READER_INFO_MODE, ucData);
```

---

#### 5.2.1.5. Data type unsigned char[] (UCHAR[])

---

```
UCHAR ucData[] = {0x01, 0x34};
int iErr = GetData(FEDM_ISC_TMP_SOFTVER_SW_REV, ucData, 2);
int iErr = SetData(FEDM_ISC_TMP_B0_REQ_DB_ADR_EXT, ucData, 2);
```

---

**5.2.1.6. Data type int**

---

int as a data type cannot be directly supported because data type BOOL is already defined as int. Instead, the cast operator UINT or unsigned int must be placed in front.

```
int iData = 0;
int iErr = GetData(FEDM_ISC_TMP_B0_RSP_DB_EXT_ADR_E, (UINT*)&iData);
int iErr = SetData(FEDM_ISC_TMP_B0_REQ_DB_ADR_EXT, (UINT)iData);
```

---

**5.2.1.7. Data type unsigned int (UINT)**

---

```
UINT uiData = 0;
int iErr = GetData(FEDM_ISC_TMP_B0_RSP_DB_EXT_ADR_E, &uiData);
int iErr = SetData(FEDM_ISC_TMP_B0_REQ_DB_ADR_EXT, uiData);
```

---

**5.2.1.8. Data type \_\_int64**

---

```
__int64 i64Data = 0;
int iErr = GetData(FEDM_ISC_TMP_B0_RSP_DB_EXT_ADR_E, &i64Data);
int iErr = SetData(FEDM_ISC_TMP_B0_REQ_DB_ADR_EXT, i64Data);
```

---

**5.2.1.9. Data type CString resp. AnsiString (VC++, C++Builder) and STL-string**

---

ALL data that are read with a string method (CString, AnsiString, string) are hex strings. This means for example that the numerical value 159 is not sent as "159" but rather as "9F". FEDM-compatible string values thus always consist of an even number of characters. To convert string values into other data types or the reverse, use the function collection in FEDM\_Functions.

To convert numerical values into strings which as in the above example make a „159“, you should use the functions from the ANSI C library (e.g., sprintf and, for the other direction, sscanf).

```
CString sBusAdr;
int iErr = GetData(FEDM_ISC_TMP_INP_STATE, sBusAdr);
int iErr = SetData(FEDM_ISC_TMP_READ_CFG_MODE, sBusAdr);

string sStlBusAdr;
int iErr = GetData(FEDM_ISC_TMP_INP_STATE, sStlBusAdr);
int iErr = SetData(FEDM_ISC_TMP_READ_CFG_MODE, sStlBusAdr);
```

---

**5.2.1.10. Direct, addressed access**

---

In some programming cases (e.g., looping, copying operations) the access parameters are too static. The solution is to use a second collection of parameterizable GetData and SetData methods. These methods are supported by useful help functions in FEDM\_Functions (s. [7.1. FEDM Functions](#)).

Supported data types are UCHAR, UCHAR[], UINT, \_\_int64 and CString or AnsiString. Their use is analogous to the examples shown above.

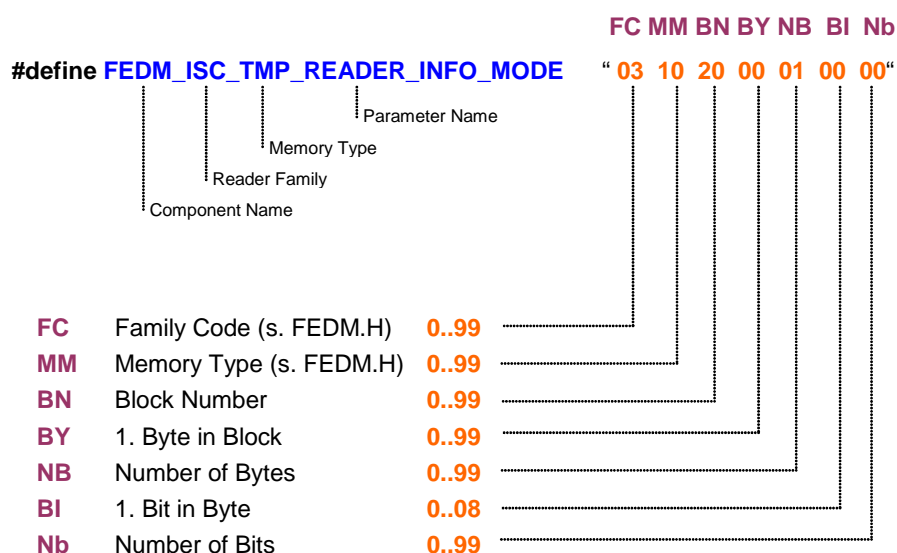
```
CHAR ucBusAdr = 0;
int iMemID = FEDM_GetMemIDofID(FEDM_ISC_TMP_SOFTVER);
```

```
int iAdr = FEDM_GetAdrOfID(FEDM_ISC_TMP_SOFTVER);
int iErr = GetData(iAdr, ucVersionInfo, 11, iMemID);
```

### 5.3. Access constants for temporary protocol data

The access constants play a central role in the data traffic between the application program and data containers for temporary protocol data in the class library, as well as within the class library between protocol method and data containers. They both identify the parameter and contain the storage location (in coded form) in one of the data containers.

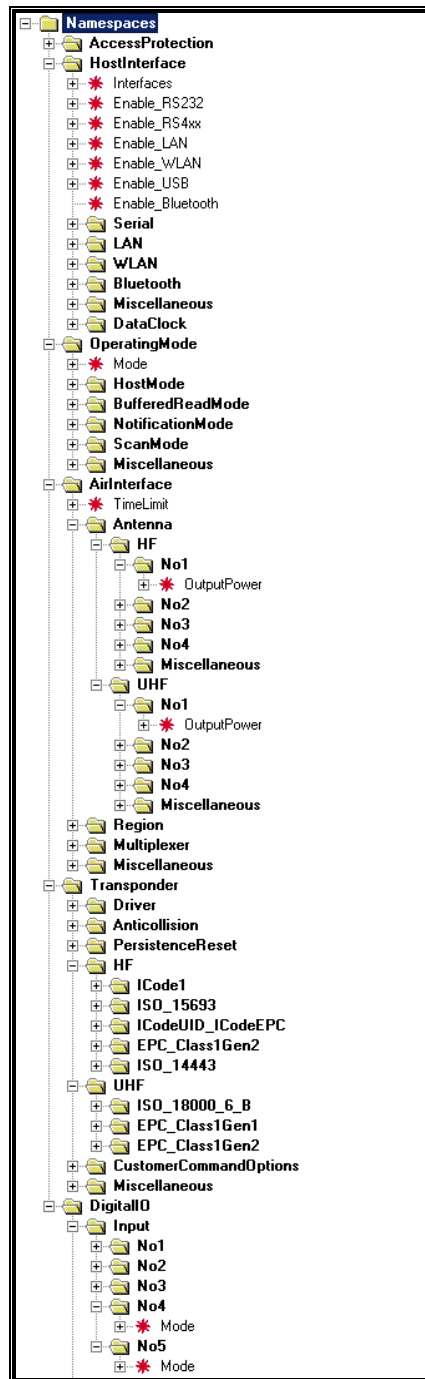
An access constant is a string and generally is structured as follows:



These access constants are used only with the methods `SetData` and `GetData`. The access constant says nothing about the data type of a protocol parameter. This is determined only by the data type of the access method. This means in the example above you could read the bus address shown either as an integer or a string or another plausible data type (s. [5.2.1. Data exchange](#)).

## 5.4. Namespaces for reader configuration parameters

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 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 namespaces

## 5.5. Tables

---

Some Reader families support protocols that can transport data for multiple transponders (e.g. ISO commands for the OBID *i-scan*® Reader family) or some sequential actions with multiple transponders for another Reader family result in a data content (e.g. data from LogDataKeys in the megalock product line) which make storage in the containers impossible. Ideally, this data is stored in the structured form of a table. Although these tables are not included in the basic classes, the basic classes nevertheless use the abstract methods `GetTableData`, `SetTableData` and `FindTableIndex` for the exchange of table data as well as the methods `GetTableSize`, `SetTableSize`, `GetTableLength` and `ResetTable` to support the administration of tables implemented in Reader classes. The use of these methods is therefore unified for all Reader families.

Access to data from a table contained in the Reader class using the methods `SetTableData` and `GetTableData` is done using unique constants like in `SetData` and `GetData`, but they do not represent a string and therefore do not contain any place coding.

Instead, the constant for the table type, the table index and the constant for the table value enables unique identification of a table value.

Example:

```
int iErr = GetTableData( int iIdx, UINT uiTableID, UINT uiDataID, ...)
```

Data types supported are `bool`, `UCHAR`, `UINT`, `__int64` and `CString` resp. `AnsiString` and `STL-string`.

Two additional `Set/GetTableData` methods are provided for block access to transponder data and support the data types `UCHAR[]`, `CString` resp. `AnsiString` and `STL-string`.

---

## 5.6. Protocol traffic

---

Protocol traffic is generated with the method `SendProtocol`. This gets only the control byte of the desired protocol. All the data necessary for the protocol transfer are taken from the data containers resp. tables. You must therefore ensure that all the protocol data have first been updated. `SendProtocol` is contained in `FEDM_DataBase` as an abstract method and is implemented only in the reader class.

Most of the reader families are bus-compatible and require the bus address in the protocol. This should be set using the method `SetBusAddress` of the `FEDM_DataBase` class.

Implementation of `SendProtocol` invokes DLL functions that always require a reader handle as the first parameter. This reader handle must be initialized using the method `SetReaderHnd` before first invoking `SendProtocol`.



---

## 5.7. Initialization methods

---

Before using the protocol method for the first time, some initializing must be performed:

- |                     |  |
|---------------------|--|
| 1. Bus address      | The bus address for the Reader is preset in the class for 255. To set a different address, use the SetBusAddress method.   |
| 2. ReaderHandle     | The handle of a Reader object in a FExxx-function library must always be stored in an instance of the Reader class using the SetReaderHnd method.  |
| 3. PortHandle       | The handle for an interface that was opened with FECOM, FETCP or FEUSB must be stored in the Reader object of a FExxx-function library. This can be done either by creating the Reader object using FExxx_NewReader or after the fact by using the method SetPortHnd. Making the change after the fact is also always possible if you need to change the port during run time. |
| 4. Language support | Error texts can be invoked in several languages. You can set the language using the SetLanguage method. The preset is for English.   |
| 5. Table size       | Some reader classes do not automatically set the size of an internal table. To make this setting, use the SetTableSize method.   |
| 6. Other            | Some Reader classes have additional initialization methods. These are described in the documentation for the respective Reader class.  |

---

## 5.8. Serializing

---

The two integrated serializing methods allow you to save data from the data containers into a file or to load data from a file into data containers:

```
int Serialize(bool bRead, char* sFileName);
```

```
int Serialize(CArchive& ar, int iMemType);
```

The first and most important version of serialize supports XML data format. One invoking serializes the entire reader configuration. Knowledge of the classes FEDM\_XMLBase and FEDM\_XMLReaderCfgDataModul is not necessary.

The second version is only suited for MFC-based applications. Each time serialize is invoked, only the contents of a data container can be serialized. Knowledge of the MFC class CArchive is presumed. This method is no longer available for the OBID i-scan® family.

---

## 5.9. Error handling

---

Nearly all the methods of the class library carry out internal error checking and return a negative value when an error is discovered. The error codes for the ID FEDM Class Library and the OBID® function libraries are organized into sectors such that they cannot overlap. The following ranges are reserved for the ID FEDM Class Library and the OBID® function libraries:

Library	Value range for error codes
ID FEDM	-101 ... -999
ID FECOM	-1000...-1099
ID FEUSB	-1100...-1199
ID FETCP	-1200...-1299
ID FERW	-2000...-2099
ID FERWA	-3000...-3099
ID FETRI	-3100...-3199
ID FEISC	-4000...-4099
ID FEFU	-4199...-4100
ID FETCL	-4299...-4200

The method `GetErrorText` for the reader class can open an English error text for the error code. The sent error code can also come from the range of an OBID® function library.

The last error code is stored in the data container `m_TmpData` and can be obtained using the member method `GetLastError`.

---

## 5.10. Language support

---

Error texts can be output in several languages. This setting is made using the `SetLanguage` method. The following languages are currently supported:

Language	Handover parameter
German	7
English (default)	9

When an error text is invoked from an OBID® function library, the handover from this text is in the defined language.

---

## 6. Class description

---

---

### 6.1. FEDM\_Base

---

The base class FEDM\_Base represents the base class of the ID FEDM C++ Class Library. It contains only methods for the derived classes. Reader communication with an instance from this class is not possible.

The header file FEDM\_Base.h also contains the definition of the standard array FEDM\_BYTE\_ARRAY, which is used for all data containers.

---

#### 6.1.1. Methods (public)

---

Method	Description
GetLibVersion	Returns in a C-string the version number of the library.
GetData	The executing (overlaid) method for reading a parameter value from a data container. This method can only be used by the derived class FEDM_DataBase or the reader class likewise derived from it.
SetData	The executing (overlaid) method for writing a parameter value to a data container. This method can only be used by the derived class FEDM_DataBase or the reader class likewise derived from it.
GetLanguage	Returns the constant for the language.
SetLanguage	Sets the language for error strings.
GetFeComFunction	The function pointer for the library FECOM is returned. If the library was not yet incorporated into the address space, it is loaded and remains in the address space. In the destructor the DLL is then again unloaded. <b>IMPORTANT: The pre-processor definition FEDM_COM_SUPPORT must be used!</b>
GetFeUsbFunction	The function pointer for the library FEUSB is returned. If the library was not yet incorporated into the address space, it is loaded and remains in the address space. In the destructor the DLL is then again unloaded. <b>IMPORTANT: The pre-processor definition FEDM_COM_SUPPORT must be used!</b>
GetFeTcpFunction	The function pointer for the library FETCP returned. If the library was not yet incorporated into the address space, it is loaded and remains in the address space. In the destructor the DLL is then again unloaded. <b>IMPORTANT: The pre-processor definition FEDM_COM_SUPPORT must be used!</b>

## 6.2. FEDM\_DataBase

The FEDM\_DataBase class is a base class derived from FEDM\_Base that provides data containers and overlaid access methods corresponding to the data types in the data containers. The size of the data containers is 0. The size is specified in the reader class derived from FEDM\_DataBase.

In addition the class contains a number of abstract methods that must be implemented in each reader class. Abstract methods permit writing of algorithms which are independent of the reader type if you invoke the methods of FEDM\_DataBase instead of the reader class methods having the same name. This also results in a type-independent interface.

### 6.2.1. Attribute (public)

Attribute	Description	Reader family	Organisation
FEDM_BYTE_ARRAY m_RFC_EEData	For reader configuration parameters	ISC – RO - RW – RWA – TRI	16 bytes per block
FEDM_BYTE_ARRAY m_RFC_RAMData	For temporary reader configuration parameters	ISC	16 bytes per block
FEDM_BYTE_ARRAY m_ACC_EEData	For reader configuration parameters	ISC	32 bytes per block
FEDM_BYTE_ARRAY m_ACC_RAMData	For temporary reader configuration parameters	ISC	32 bytes per block
FEDM_BYTE_ARRAY m_TmpData	For general temporary protocol data	ISC – RO - RW – RWA - TRI	32 bytes per block
FEDM_BYTE_ARRAY m_MjpData	For temporary protocol data of a multijob-Poll	RWA – TRI	16 bytes per block
FEDM_BYTE_ARRAY m_SN_Mem	For transponder serial numbers	RW – RWA - TRI	16 bytes per block
FEDM_BYTE_ARRAY m_ID_Mem	For transponder ID numbers	RW – RWA	16 bytes per block
FEDM_BYTE_ARRAY m_AC_Mem	For transponder account data	RWA	16 bytes per block
FEDM_BYTE_ARRAY m_PubMem	For transponder public data blocks	RW – RWA – TRI	4 bytes per block 16 Byte each Block for RWA
FEDM_BYTE_ARRAY m_SecMem	For transponder secret data blocks	RW – RWA	4 bytes per block
FEDM_BYTE_ARRAY m_ConfMem	For transponder configuration data	RW – RWA – TRI	4 bytes per block
FEDM_BYTE_ARRAY m_DateMem	For transponder date data	RWA - TRI	16 bytes per block

### 6.2.2. Methods (public)

Method	Description
SetReaderHnd	Sets the handle obtained from the OBID reader-specific function library (FEISC, FERO, FERW, FERWA, FETRI, ...).
GetReaderHnd	Returns the handle obtained from the OBID reader-specific function library (FEISC, FERO, FERW, FERWA, FETRI, ...).
SetBusAddress	Sets the bus address for protocol traffic.
GetBusAddress	Gets the bus address of the protocol traffic.
GetFamilyCode	Gets the short string for the reader family classification.
GetReaderName	Gets the short string of the reader name..
GetReaderType	Gets the reader type.
Serialize	Sub-method for implementing the XML interface.
SetData	<p>The central (overlaid) method for writing a parameter value to a data container. The invocation is passed to the base class FEDM_Base after the data container type (memory type constant) has been determined from the access constant. SetData supports the following data types: bool, BOOL, UCHAR, UCHAR-Array, UNIT, __int64, CString resp. AnsiString, STL-string and C-string.</p> <p>A second variation allows read access by specifying the exact index and memory type constant. This variation supports the following data types: UCHAR, UCHAR-Array, UNIT, __int64 and CString resp. AnsiString.</p>
GetData	<p>The central (overlaid) method for reading a parameter value from a data container. The invocation is passed to the base class FEDM_Base after the data container type (memory type constant) has been determined from the access constant. GetData supports the following data types: bool, BOOL, UCHAR, UCHAR-Array, UNIT, __int64, CString resp. AnsiString, STL-string and C-string.</p> <p>A second variation allows write access by specifying the exact index and memory type constant. This variation supports the following data types: UCHAR, UCHAR-Array, UNIT, __int64 and CString resp. AnsiString.</p>

### 6.2.3. Abstract methods (public)

Method	Description
<i>EvalLibDependencies</i>	Method verifies the compatibility with dependent function libraries
<i>SendProtocol</i>	The central communication method.
<i>FindBaudRate</i>	Detects the baudrate und protocol frame of the reader and adjusts the serial port.
<i>GetLastProt</i>	Method for getting the last send or receive protocol.
<i>SetPortHnd</i>	Transfers the handle obtained from the function library contained in the protocol transfer layer (FECOM.DLL, FEUSB.DLL, ...) to the function library contained in the OBID reader-specific protocol layer (FEISC.DLL, FERW.DLL, FERWA.DLL, FETRI.DLL, ...).
<i>GetPortHnd</i>	Gets the handle obtained from the function library contained in the protocol transfer layer (FECOM.DLL, FEUSB.DLL, ...) from the function library contained in the OBID reader-specific protocol layer (FEISC.DLL, FERW.DLL, FERWA.DLL, FETRI.DLL, ...).
<i>SetProtocolFrameSupport</i>	Selects the protocol type for communication with the reader
<i>GetProtocolFrameSupport</i>	Queries the protocol type.

Method	Description
<i>SetReaderType</i>	Method gets the ID of the reader type as a parameter and sets internal environment variables for this reader type.
<i>GetLastError</i>	Returns the last error code.
<i>GetLastStatus</i>	Returns the status value of the last protocol.
<i>GetErrorText</i>	Returns the text associated with the error code.
<i>GetStatusText</i>	Returns the text corresponding to each status byte.
<i>Serialize</i>	Main method for serializing. Allows serializing of the container data in files. Two versions are implemented: one version for file type XML and a second version for MFC-based applications.
<i>SerializeIn</i>	Sub-method for implementing the XML interface.
<i>SerializeOut</i>	Sub-method for implementing the XML interface.
<i>GetTableData</i>	The central (overlaid) method for reading a parameter value or data blocks from a table.  This variation supports the following data types: bool, UCHAR, UCHAR-Array, UINT, __int64, CString resp. AnsiString and STL-string.
<i>SetTableData</i>	The central (overlaid) method for writing a parameter value or data blocks to a table.  This variation supports the following data types: bool, UCHAR, UCHAR-Array, UINT, __int64, CString resp. AnsiString and STL-string.
<i>FindTableIndex</i>	The central (overlaid) method for getting the table index based on a value.  This variation supports the following data types: UCHAR, UCHAR-Array, UINT, __int64, CString resp. AnsiString..
<i>VerifyTableData</i>	Equation of received with sent transponder data (only OBID i-scan® family)
<i>GetTableSize</i>	Gets the size of a table.
<i>SetTableSize</i>	Sets the size of a table.
<i>GetTableLength</i>	Gets the number of valid table entries.
<i>ResetTable</i>	Initializes the complete table or a single member of it.

---

### 6.3. FEDM\_XMLBase

---

Class FEDM\_XMLBase is the base class for serializing object data in XML format. It contains all the base methods, a series of attributes and the object tree for the XML structure.

FEDM\_XMLBase is not a general XML class for any file. Rather, this class is especially tailored for the specialized task of serializing reader configurations or similar object data. This also means it is small compared with the powerful commercial or non-commercial XML libraries.

**IMPORTANT:** For including the XML serialization classes, the pre-processor definition **\_FEDM\_XML\_SUPPORT** must be set!

---

#### 6.3.1. Methods (public)

---

Method	Description
OpenDoc	Opens an XML document
CloseDoc	Closes an XML document
ReadDoc	Reads the complete contents of the XML document
WriteDoc	Writes the Unicode strings to the document
IsXmlDoc	Checks whether the file opened with OpenDoc is an XML document
HasOBIDTag	Checks whether the file opened with OpenDoc contains an OBID tag
BuildTag	Builds a new tag structure
AddTagValue	Adds the contents to a tag
AddTagAttrib	Adds an attribute with value to a tag
AddTagItem	Inserts the tag structure generated with BuildTag into the internal XML tree
FindTag	Looks for a tag in the internal XML tree
GetTagValue	Returns the contents of a tag
GetTagAttrib	Returns an attribute with value of a tag
GetLastError	Returns the contents of the internal error variable m_iLastError

## 6.4. FEDM\_XMLReaderCfgDataModul

Class FEDM\_XMLReaderCfgDataModul is derived from the base class FEDM\_XMLBase and has been especially developed for serializing reader configuration data from the data containers m\_RFC\_EEData and m\_RFC\_RAMData implemented in the reader class FEDM\_DataBase.

### 6.4.1. Methods (public)

Method	Description
Serialize	Sub-method reads entire XML document and saves the reader configuration data in the data containers of the reader class FEDM_DataBase.
SerializeOut	Sub-method writes entire XML document with header and reader configuration data from the data containers of the reader class FEDM_DataBase.
QueryDocType	Opens an XML document and gets the document type (currently only „Reader Configuration File“).
QueryDocVersion	Opens an XML document and gets the document version (e.g. „1.0“).
QueryReaderFamily	Opens an XML document and gets the reader family from the „reader family“ tag
QueryReaderType	Opens an XML document and gets the reader type from the „reader-type“ tag.
GetComment	Returns the comment text from the variable m_wsComment.
SetComment	Saves the comment text in the variable m_wsComment. If there is no comment text, this is written in SerializeOut to the XML document.
GetPrgName	Returns the program name from the variable m_wsPrgName.
SetPrgName	Saves the program name in the variable m_wsPrgName. If there is no program name, this is written in SerializeOut to the XML document.
GetPrgVer	Returns the program version from the variable m_wsPrgVer.
SetPrgVer	Saves the program version in the variable m_wsPrgVer. If there is no program version, this is written in serializeOut to the XML document.
GetHost	Returns the TCP/IP host address from the variable m_wsHost. The TCP/IP host address is inserted into the XML file of the ISOStart program.
GetPort	Returns the port number from the variable m_wsPort. The communication mode („Serial“, „USB“, „TCP“) is inserted into the XML file of the ISOStart program.
GetCommMode	Returns the communication mode from the variable m_wsCommMode. The communication mode is inserted into the XML file of the ISOStart program.



## 7. Global functions

### 7.1. FEDM\_Functions

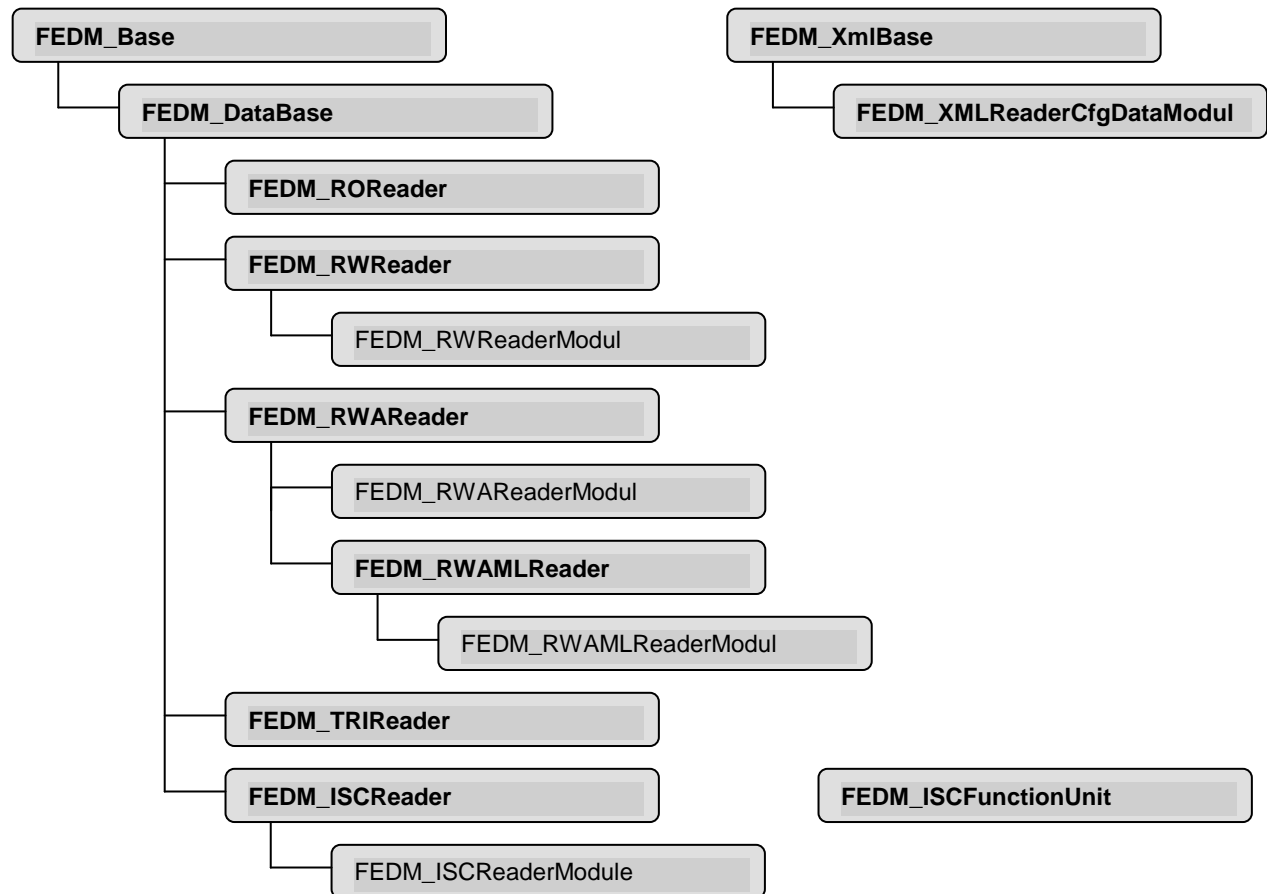
Function	Description
FEDM_GetMemIDOfID	Gets the memory type constant from the access constant. (s. <a href="#">7.1. FEDM_Functions</a> ). Example: FEDM_GetMemIDOfID(FEDM_ISCM_EE_COM_BUSADR) returns the value 3.
FEDM_GetAdrOfID	Gets the address of a parameter from the access constant.
FEDM_GetByteCntOfID	Gets the number of bytes that a parameter consists of from the access constant.
FEDM_ToRAM	Access to parameters in the container m_RFC_RAMData with the access constant for the container m_RFC_EEData.
FEDM_MdfyMemID	Modifies the memory type constant of the access constant
FEDM_MdfyBlockNr	Modifies the block number of the access constant.
FEDM_AddOff2BlockNr	Adds an offset to the block number of the access constant.
FEDM_ConvHexCharToInt	Converts a C-string with a hex string into an integer value. The string is allowed to contain a maximum of 8 characters. All characters except 0..9, a..f, A..F are removed. Example: sln = "1122F05E" -> iOut = 287502430
FEDM_ConvHexCharToUInt	Converts a C-string with a hex string into an unsigned integer value. The string is allowed to contain a maximum of 8 characters. All characters except 0..9, a..f, A..F are removed. Example: sln = "1122F05E" -> uiOut = 287502430
FEDM_ConvHexCharToInt64	Converts a C-string with a hex string into a 64-bit integer value. The string is allowed to contain a maximum of 16 characters. All characters except 0..9, a..f, A..F are removed. Example: sln = "1122F05E1122F05E" -> i64Out = 1234813534658031710
FEDM_ConvHexCharToUChar	Converts a C-string with a hex string into a C-string. All characters except 0..9, a..f, A..F are first removed. Example: sln = "1122F05E" --> ucOutBuf = {0x11, 0x22, 0xF0, 0x5E}
FEDM_ConvHexStrToInt	Converts a hex string into an integer value. The string is allowed to contain a maximum of 8 characters. All characters except 0..9, a..f, A..F are removed. Example: sln = "1122F05E" -> iOut = 287502430
FEDM_ConvHexStrToUInt	Converts a hex string into an unsigned integer value. The string is allowed to contain a maximum of 8 characters. All characters except 0..9, a..f, A..F are removed. Example: sln = "1122F05E" -> uiOut = 287502430
FEDM_ConvHexStrToInt64	Converts a hex string into a 64-bit integer value. The string is allowed to contain a maximum of 16 characters. All characters except 0..9, a..f, A..F are removed. Example: sln = "1122F05E1122F05E" -> i64Out = 1234813534658031710
FEDM_ConvHexStrToUChar	Converts a hex string into a C-string. All characters except 0..9, a..f, A..F are first removed. Example: sln = "1122F05E" --> ucOutBuf = {0x11, 0x22, 0xF0, 0x5E}
FEDM_ConvHexUCharToInt	Converts a C-string into an integer value. The C-string is allowed to contain a maximum of 4

Function	Description
	characters. Example: uclnBuf = {0x11, 0x22, 0xF0, 0x5E} -> iOut = 287502430
FEDM_ConvHexUCharToUInt	Converts a C-string into an n unsigned integer value. The C-string is allowed to contain a maximum of 4 characters. Example: uclnBuf = {0x11, 0x22, 0xF0, 0x5E} -> uiOut = 287502430
FEDM_ConvHexUCharToInt64	Converts a C-string into a 64-bit integer value. The C-string is allowed to contain a maximum of 8 characters. Example: uclnBuf = {0x11, 0x22, 0xF0, 0x5E, 0x11, 0x22, 0xF0, 0x5E } -> iOut = 1234813534658031710
FEDM_ConvHexUCharToHexChar	Converts a C-string into a C-string with a hex string. Example: uclnBuf = {0x11, 0x22, 0xF0, 0x5E} -> sOut = "1122F05E"
FEDM_ConvHexUCharToHexStr	Converts a C-string into a hex string. Example: uclnBuf = {0x11, 0x22, 0xF0, 0x5E} -> sOut = "1122F05E"
FEDM_ConvHexUCharToBcdChar	Converts an Byte-Array into a BCD-Array. Example: uclnBuf = {0x02, 0x01, 0x00, 0x00, 0x01, 0x04} -> ucOutBuf = {0x21, 0x00, 0x14}
FEDM_ConvBcdCharToHexUChar	Converts a BCD-Array into an Byte-Array. Example: uclnBuf = {0x21, 0x00, 0x14} -> ucOutBuf = {0x02, 0x01, 0x00, 0x00, 0x01, 0x04}
FEDM_ConvIntToHexStr	Converts an integer value into a hex string. Example: iln = 287502430 -> sOut = "1122F05E"
FEDM_ConvIntToHexUChar	Converts an integer value into a C-string. Example: iln = 287502430 -> ucOutBuf = {0x11, 0x22, 0xF0, 0x5E}
FEDM_ConvIntToHexChar	Converts an integer value into a C-string with a hex string. Example: iln = 287502430 -> sOut = "1122F05E"
FEDM_ConvUIntToHexStr	Converts an unsigned integer value into a hex string. Example: uiln = 287502430 -> sOut = "1122F05E"
FEDM_ConvUIntToHexUChar	Converts an unsigned integer value into a C-string. Example: uiln = 287502430 -> ucOutBuf = {0x11, 0x22, 0xF0, 0x5E}
FEDM_ConvUIntToHexChar	Converts an unsigned integer value into a C-string with a hex string. Example: uiln = 287502430 -> sOut = "1122F05E"
FEDM_ConvInt64ToHexStr	Converts a 64-bit integer value into a hex string. Example: i64ln = 1234813534658031710 -> sOut = "1122F05E1122F05E"
FEDM_ConvInt64ToHexUChar	Converts a 64-bit integer value into a C-string. Example: i64ln = 1234813534658031710 -> iOutBuf = {0x11, 0x22, 0xF0, 0x5E, 0x11, 0x22, 0xF0, 0x5E }
FEDM_ConvInt64ToHexChar	Converts a 64-bit integer value into a C-string with a hex string. Example: i64ln = 1234813534658031710 -> sOut = "1122F05E1122F05E "
FEDM_ConvTwoAsciiToUChar	Converts two ASCII characters (0..9, a..f, A..F) into a char-value.
FEDM_RemNoHexChar	Removes all non-ASCII characters from a string and copies the results to a C-string.
FEDM_IsHex	Tests a string for ASCII characters (0..9, a..f, A..F).



## 8. Appendix

### 8.1. Class Tree



## 8.2. List of error codes

All listed error codes are located in the file FEDM.h.

Error constant	Value	Description
FEDM_MODIFIED	1	Indicates a modification of a container. This is not an error.
FEDM_OK	0	No error
FEDM_ERROR_BLOCK_SIZE	-101	Block size in the access constant is incorrect
FEDM_ERROR_BIT_BOUNDARY	-102	Bit boundary in the access constant is incorrect
FEDM_ERROR_BYTE_BOUNDARY	-103	Byte boundary in the access constant is incorrect
FEDM_ERROR_ARRAY_BOUNDARY	-104	Array boundary of a data container was exceeded
FEDM_ERROR_BUFFER_LENGTH	-105	Length of the data buffer is insufficient
FEDM_ERROR_PARAMETER	-106	Unknown transfer parameter
FEDM_ERROR_STRING_LENGTH	-107	Transferred string is too long
FEDM_ERROR_ODD_STRING_LENGTH	-108	Transferred string contains an odd number of characters
FEDM_ERROR_NO_DATA	-109	No data in the protocol
FEDM_ERROR_NO_READER_HANDLE	-110	No reader handle set
FEDM_ERROR_NO_PORT_HANDLE	-111	No port handle set
FEDM_ERROR_UNKNOWN_CONTROL_BYTE	-112	Unknown control byte
FEDM_ERROR_UNKNOWN_MEM_ID	-113	Unknown memory ID
FEDM_ERROR_UNKNOWN_POLL_MODE	-114	Unknown poll mode
FEDM_ERROR_NO_TABLE_DATA	-115	No data in a table
FEDM_ERROR_UNKNOWN_ERROR_CODE	-116	Unknown error code
FEDM_ERROR_UNKNOWN_COMMAND	-117	Unknown command
FEDM_ERROR_UNSUPPORTED	-118	No support for this parameter or function
FEDM_ERROR_NO_MORE_MEM	-119	No more program memory available
FEDM_ERROR_NO_READER_FOUND	-120	No reader found
FEDM_ERROR_NULL_POINTER	-121	The transferred pointer is NULL
FEDM_ERROR_UNKNOWN_READER_TYPE	-122	Unknown reader type
FEDM_ERROR_UNSUPPORTED_READER_TYPE	-123	The Function doesn't support this reader type
FEDM_ERROR_UNKNOWN_TABLE_ID	-124	Unknown table constant
FEDM_ERROR_UNKNOWN_LANGUAGE	-125	Unknown language constant

Error constant	Value	Description
FEDM_ERROR_NO_TABLE_SIZE	-126	The table has the size 0
FEDM_ERROR_SENDBUFFER_OVERFLOW	-127	The Sendbuffer is full
FEDM_ERROR_VERIFY	-128	Data are not equal
FEDM_ERROR_OPEN_FILE	-129	File open error
FEDM_ERROR_SAVE_FILE	-130	File save error
FEDM_ERROR_UNKNOWN_TRANSPONDER_TYPE	-131	Unknown transponder type
FEDM_ERROR_READ_FILE	-132	Read file error
FEDM_ERROR_WRITE_FILE	-133	Write file error
FEDM_ERROR_UNKNOWN_EPC_TYPE	-134	Unknown EPC-Type
FEDM_ERROR_UNSUPPORTED_PORT_DRIVER	-135	Function does not support the active communication driver
FEDM_ERROR_UNKNOWN_ADDRESS_MODE	-136	Unknown address mode
FEDM_ERROR_ALREADY_CONNECTED	-137	Reader object is already connected with a communication port
FEDM_ERROR_NOT_CONNECTED	-138	Reader object is not connected with a communication port
FEDM_ERROR_NO_MODULE_HANDLE	-139	No module handle found
FEDM_ERROR_EMPTY_MODULE_LIST	-140	The module list is empty
FEDM_ERROR_MODULE_NOT_FOUND	-141	Module not found in module list
FEDM_ERROR_DIFFERENT_OBJECTS	-142	Runtime objects are different
FEDM_ERROR_NOT_AN_EPC	-143	IDD of transponder is not an EPC
FEDM_ERROR_OLD_LIB_VERSION	-144	Old library file (error code for Java/.NET-Libraries)
FEDM_ERROR_WRONG_READER_TYPE	-145	Wrong reader type
FEDM_ERROR_CRC	-146	CRC error in file
FEDM_ERROR_CFG_BLOCK_PREVIOUSLY_NOT_READ	-147	Configuration block must be read first
FEDM_ERROR_UNSUPPORTED_CONTROLLER_TYPE	-148	Unsupported controller type
FEDM_ERROR_VERSION_CONFLICT	-149	Version conflict with one or more dependent libraries
FEDM_ERROR_UNSUPPORTED_NAMESPACE	-150	The namespace is not supported by the reader type
FEDM_ERROR_TASK_STILL_RUNNING	-151	Asynchronous task is still running
FEDM_ERROR_TAG_HANDLER_NOT_IDENTIFIED	-152	TagHandler type could not be identified
FEDM_ERROR_UNVALID_IDD_LENGTH	-153	Value of IDD-Length is out of range
FEDM_ERROR_UNVALID_IDD_FORMAT	-154	Value of IDD-Format is out of range
FEDM_ERROR_UNKNOWN_TAG_HANDLER_TYPE	-155	Unknown TagHandler type
FEDM_ERROR_UNSUPPORTED_TRANSPONDER_TYPE	-156	Transponder- or Chip-Type is not supportet
FEDM_ERROR_CONNECTED_WITH_OTHER_MODULE	-157	Only TCP/IP: a connection to the same Reader still established by another Reader

Error constant	Value	Description
		module.
FEDM_ERROR_INVENTORY_NO_TID_IN_UID	-158	Inventory with return of UID = EPC + TID, but TID is missing
FEDM_XML_ERROR_NO_XML_FILE	-200	File is not a XML document
FEDM_XML_ERROR_NO_OBID_TAG	-201	File contains no element 'OBID'
FEDM_XML_ERROR_NO_CHILD_TAG	-202	No sub-element found
FEDM_XML_ERROR_TAG_NOT_FOUND	-203	Element not in the document
FEDM_XML_ERROR_DOC_NOT_WELL_FORMED	-204	XML document not well-formed
FEDM_XML_ERROR_NO_TAG_VALUE	-205	No content of element found
FEDM_XML_ERROR_NO_TAG_ATTRIBUTE	-206	No attribute found
FEDM_XML_ERROR_DOC_FILE_VERSION	-207	Invalid document version
FEDM_XML_ERROR_DOC_FILE_FAMILY	-208	The Document is for another reader family
FEDM_XML_ERROR_DOC_FILE_TYPE	-209	Wrong file type
FEDM_XML_ERROR_WRONG_CONTROLLER_TYPE	-210	Wrong controller type
FEDM_XML_ERROR_WRONG_MEM_BANK_TYPE	-211	Wrong memory bank

---

### 8.3. List of reader families

---

All listed constants are located in the file FEDM.h.

Reader constant	Values	Description
FEDM_RW_FAMILY	1	
FEDM_RWA_FAMILY	2	
FEDM_ISC_FAMILY	3	
FEDM_TRI_FAMILY	4	
FEDM_RO_FAMILY	5	

---

### 8.4. List of protocol type constants

---

All listed constants are located in the file FEDM.h.

Protokolltyp-Konstante	Wert	Beschreibung
FEDM_PRT_FRAME_STANDARD	1	Voreinstellung
FEDM_PRT_FRAME_ADVANCED	2	

---

### 8.5. List of language constants

---

All listed constants are located in the file FEDM.h.

Language constant	Value	Description
FEDM_LANG_GERMAN	7	
FEDM_LANG_ENGLISH	9	preset

---

### 8.6. List of memory type constants

---

All listed constants are located in the file FEDM.h.

Memory type constant	Value	Description
FEDM_RFC_EEDATA_MEM	3	for reader configuration parameters
FEDM_RFC_RAMDATA_MEM	4	for temporary reader configuration parameters
FEDM_ACC_EEDATA_MEM	5	for reader configuration parameters



Memory type constant	Value	Description
FEDM_ACC_RAMDATA_MEM	6	for temporary reader configuration parameters
FEDM_TMPDATA_MEM	10	for general temporary protocol data
FEDM_MJPDATA_MEM	11	for temporary protocol data of a multijob-Poll
FEDM_SN_MEM	20	for transponder serial numbers
FEDM_ID_MEM	21	for transponder ID numbers
FEDM_AC_MEM	22	for transponder account data
FEDM_PUB_MEM	23	for transponder public data blocks
FEDM_SEC_MEM	24	for transponder secret data blocks
FEDM_CONF_MEM	25	for transponder configuration data
FEDM_DATE_MEM	26	for transponder date data

---

## 8.7. Macros

---

All listed macros are located in the file FEDM.h.

Macro	Description
FEDM_CHK1	Checks whether the return value of a function is negative and returns the error code. This macro can only be used in reader class.
FEDM_CHK2	Checks whether the return value of a function is negative and, if so, sets the global error variable in FEDM_TMPDATA_MEM. This macro can only be used in reader class.
FEDM_CHK3	Checks whether a pointer is NULL.
FEDM_CHK4	Checks whether the return value of a function is not 0 and, if so, sets the global error variable in FEDM_TMPDATA_MEM. This macro can only be used in reader class.
FEDM_CHK5	Checks whether a pointer is NULL and, if so, sets the global error variable in FEDM_TMPDATA_MEM. This macro can only be used in reader class.
FEDM_CHK6	Checks whether the return value of a function is negative and returns without an error code. This macro can only be used in reader class.
FEDM_CHK7	Checks whether the return value of a function is negative and, if so, sets the global error variable in FEDM_TMPDATA_MEM and returns with NULL. This macro can only be used in reader class.
FEDM_CHK8	Checks, whether the value is null or negativ and if so, sets the global error variable to FEDM_ERROR_STRING_LENGTH. This macro can only be used in reader class.
FEDM_RETURN	Sets the global error variable in FEDM_TMPDATA_MEM. This macro can only be used in reader class.
FEDM_IS_COPPORT	Checks whether the port handle is a handle of a serial port.
FEDM_IS_USBPORT	Checks whether the port handle is a device handle of a USB port.
FEDM_IS_TCPPORT	Checks whether the port handle is a socket handle of a TCP/IP port.

## 8.8. Revision history

---

### V4.03.00

- New error code: FEDM\_ERROR\_INVENTORY\_NO\_TID\_IN\_UID

### V4.02.00

- This new version is not compatible with the former version. More update descriptions can be found in the manual of part B.
- Windows:
  1. Migration from Visual Studio 2005 to Visual Studio 2010.
  2. First release of 64-Bit version
  3. Dynamic binding to Log-Manager
- First Release for Mac OS X, V10.7.3 or higher

### V4.00.00

- This new version is not compatible with the former version. More update descriptions can be found in the manual of part B.

### V3.03.00

- This new version might not be fully compatible with the former version, dependent from the used classes, methods and constants.
- More update descriptions can be found in the manual of part B.

### V3.01.00

- This new version might not be fully compatible with the former version, dependent from the used classes, methods and constants.
- USB support for Windows CE
- New error codes

### V3.00.00

- This new version might not be fully compatible with the former version, dependent from the used classes, methods and constants.

- USB support for Linux
- Dynamic link libraries (DLL/SO) available
- New directory organisation for sources
- Only single include file
- New error codes
- The Access Constants for configuration parameters are replaced by the namespace ReaderConfig (actually only for OBID i-scan® family)
- Renaming of the following constants

Old constant	New constant
FEDM_EEDATA_MEM	FEDM_RFC_EEDATA_MEM
FEDM_RAMDATA_MEM	FEDM_RFC_RAMDATA_MEM

- Renaming of the following variables in the class FEDM\_DataBase

Old name	New name
EEData	m_RFC_EEData
RAMData	m_RFC_RAMData
TmpData	m_TmpData

#### V2.05.06

- The Linux library is compiled with GCC 3.3.3 under SuSE Linux 9.1

#### V2.05.01

- Modified licence agreement

#### V2.05.00

- New reader class FEDM\_ISCReaderModule (s. Part B of the manual collection)
- New globale functions for converting HexChar to different other data types (FEDM\_ConvHexCharTo...)

#### V2.04.00

- Optional static bindung with pre-processor definition **\_FEDM\_SUPPORT\_SLINK**

#### V2.03.00

- The blocksize of the byte array TmpData for temporary data is changed from 16 to 32.

- New pre-processor definition **\_FEDM\_XML\_SUPPORT** for including the XML serialization classes. This option was not necessary in previous versions. In front of the re-compilation of a project, this definition must be set, if the XML serialization classes are used.
- New pre-processor definition **\_FEDM\_MFC\_SUPPORT** for including the MFC classes (basically CString and CArchive). This option was not necessary in previous versions. In front of the re-compilation of a project, this definition must be set, if the MFC classes are used.

#### V2.02.00

- No changes in the base classes.

#### V2.01.00

- No changes in the base classes.

#### V2.00.00

- No changes in the base classes.

#### V1.09.10

- New error codes.
- Support for Advanced Protocol Length with two length bytes.

#### V1.08.00

- Implements new classes FEDM\_XMLBase and EDM\_XMLReaderCfgDataModul Interface for serializing reader configuration in XML format.
- FEDM\_DataBase::GetFamilyCode function was changed and is no longer compatible with the previous version.
- New functions in abstract base class FEDM\_DataBase: GetReaderName, GetReaderType, SetReaderType and three new serialize functions.
- Support for USB readers.
- Supports TCP/IP interface.
- Conversion to dynamic linking of the communications libraries (FECOM, FETCP, FEUSB) using function pointers. This results in new functions in the base class FEDM\_Base: GetFeComFunction, GetFeTcpFunction, GetFeUsbFunction
- Support for data type bool, \_\_int64 and STL-string for the functions GetTableData, SetTableData, FindTableData.
- Flexible incorporation of the communications libraries (FECOM, FETCP, FEUSB) using pre-processor definitions.

- Flexible compiling for Windows or Linux using pre-processor definitions.
- New error codes.

#### V1.06.00

- New classes FEDM\_RWAMLReader, FEDM\_RWReaderModul, FEDM\_RWAReaderModul and FEDM\_RWAMLReaderModul.
- Support of GNU C-Compiler under Linux.

#### V1.05.00

- internal version.

#### V1.04.00

- New classes for Read-Only reader family and for megalock products.

#### V1.03.00

- Language support with new functions SetLanguage und GetLanguage. German and english strings are implemented.
- Setting of table sizes during run-time with the new function SetTableSize. This function is not supported in every reader class.
- The function ResetTable has a new parameter.
- The functions GetData and SetData of the base class FEDM\_Base are now public.

#### V1.02.00

- Porting to Borland C++Builder is finished.
- Error handling in FEDM\_DataBase completed.
- Minor errors corrected, minor additions
- New abstract functions in the base class FEDM\_DataBase

#### V1.00.00

- First release version