OBID®
Welcome to RFID

HANDBUCH

C++ Klassenbibliothek

ID FEDM

Version 4.06.06

Teil A

Software-Support für OBID® Leserfamilien

Betriebssystem	Ausführung		Anmerkungen
	32-Bit	64-Bit	
Windows XP	X	(X)	bei 64-Bit nur mit 32-Bit Laufzeitsystem
Windows Vista / 7 / 8	X	X	
Windows CE	X	-	
Linux	X	X	
Android	X		Auf Anfrage
Apple Max OS X	-	X	ab V10.7.3, Architektur x86_64

final
public (B)
2014-09-26
H10102-22d-ID-B.DOC

FEIG ELECTRONIC

Hinweis

 FEIG ELECTRONIC GmbH
Lange Straße 4
D-35781 Weilburg-Waldhausen
eMail: obid-support@feig.de

Alle früheren Ausgaben verlieren mit diesem Handbuch ihre Gültigkeit.
Die Angaben in diesem Handbuch können ohne vorherige Ankündigung geändert werden.

Die Zusammenstellung der Informationen in diesem Handbuch erfolgt nach bestem Wissen und Gewissen. FEIG ELECTRONIC GmbH übernimmt keine Gewährleistung für die Richtigkeit und Vollständigkeit der Angaben in diesem Handbuch. Insbesondere kann FEIG ELECTRONIC GmbH nicht für Folgeschäden aufgrund fehlerhafter oder unvollständiger Angaben haftbar gemacht werden. Da sich Fehler, trotz aller Bemühungen nie vollständig vermeiden lassen, sind wir für Hinweise jederzeit dankbar.

FEIG ELECTRONIC GmbH übernimmt keine Gewährleistung dafür, dass die in diesem Dokument enthaltenden Informationen frei von fremden Schutzrechten sind. FEIG ELECTRONIC GmbH erteilt mit diesem Dokument keine Lizenzen auf eigene oder fremde Patente oder andere Schutzrechte.

Die in diesem Handbuch gemachten Installationsempfehlungen gehen von günstigsten Rahmenbedingungen aus. FEIG ELECTRONIC GmbH übernimmt keine Gewähr für die einwandfreie Funktion einer OBID®-Anlage in systemfremden Umgebungen.

Lizenzvertrag über die Nutzung der Software

Dies ist ein Vertrag zwischen Ihnen und der FEIG ELECTRONIC GmbH (nachfolgend "FEIG") über die Nutzung der überlassenen Klassenbibliothek ID FEDM und die vorliegende Dokumentation, nachfolgend Lizenzmaterial genannt. Mit der Installation und Benutzung der Software erklären Sie sich mit allen Bestimmungen dieses Vertrages ausnahmslos und ohne Einschränkung einverstanden. Wenn Sie mit den Bestimmungen dieses Vertrages nicht oder nicht vollständig einverstanden sind, dürfen Sie das Lizenzmaterial nicht installieren oder anderweitig benutzen. Das überlassene Lizenzmaterial ist Eigentum der FEIG ELECTRONIC GmbH und ist international urheberrechtlich geschützt.

§1 Vertragsgegenstand und Vertragsumfang

1. FEIG gewährt Ihnen das Recht, das überlassene Lizenzmaterial zu installieren und zu den nachstehenden Bedingungen zu nutzen.
2. Sie dürfen sämtliche Bestandteile des Lizenzmaterials auf einer Festplatte oder einem sonstigen Speichermedium installieren. Die Installation und Nutzung darf auch auf einem Netzwerk-Fileserver erfolgen. Sie dürfen Sicherheitskopien des Lizenzmaterials anfertigen.
3. FEIG gewährt Ihnen das Recht die dokumentierte Klassenbibliothek für die Entwicklung eigener Anwendungsprogramme oder Programmbibliotheken zu verwenden und Sie dürfen Laufzeitdateien ohne Abgabe von Lizenzgebühren vertreiben, unter der Voraussetzung, dass diese Anwendungsprogramme oder Programmbibliotheken dazu dienen, Geräte und / oder Anlagen anzusteuern oder zu betreiben, die von FEIG entwickelt und / oder vertrieben werden.

§2. Schutz des Lizenzmaterials

1. Das Lizenzmaterial ist geistiges Eigentum von FEIG und seinen Lieferanten. Es ist gemäß Urheberrecht, internationalen Verträgen und einschlägigen Gesetzen des Landes geschützt, in dem sie genutzt wird. Struktur, Organisation und Code der Software sind wertvolles Geschäftsgeheimnis und vertrauliche Information von FEIG und seinen Lieferanten.
2. Sie verpflichten sich, die Klassenbibliothek sowie die Dokumentation nicht zu ändern.
3. Soweit FEIG im Lizenzmaterial Schutzvermerke, wie Copyright-Vermerke und andere Rechtsvorbehalte angebracht hat, sind Sie verpflichtet, diese unverändert beizubehalten sowie in alle von Ihnen hergestellten vollständigen oder teilweisen Kopien in unveränderter Form zu übernehmen.
4. Die Weitergabe von Lizenzmaterial ist weder vollständig noch auszugsweise gestattet, solange dazu keine explizite anderslautende Vereinbarung zwischen Ihnen und FEIG getroffen wurde. Nicht betroffen von dieser Regelung sind solche Anwendungsprogramme oder Programmbibliotheken, die gem. §1 Absatz 3. dieser Vereinbarung erstellt und vertrieben werden, solange die Weitergabe des Sourcecodes der Klassenbibliothek ausgeschlossen bleibt.

§3 Gewährleistung und Haftungsbeschränkungen

1. Sie stimmen mit FEIG darüber überein, dass es nicht möglich ist, EDV-Programme so zu entwickeln, dass sie für alle Anwendungsbedingungen fehlerfrei sind. FEIG weist Sie ausdrücklich darauf hin, dass die Installation eines neuen Programms bereits vorhandene Software beeinflussen kann, und zwar auch solche Software, die nicht gleichzeitig mit der neuen Software ausgeführt wird. FEIG haftet in keinem Fall für direkte oder indirekte Schäden, für Folgeschäden oder Sonderschäden, Einschließlich entgangenen Geschäftsgewinn oder entgangener Einsparungen. Wenn Sie sicherstellen wollen, dass es zu keinerlei Beeinflussung eines bereits installierten Programms kommt, dürfen Sie die vorliegende Software nicht installieren.
2. FEIG weist ausdrücklich darauf hin, dass mit der Software irreversible Einstellungen und Anpassungen an Geräten vorgenommen werden können, wodurch diese Geräte zerstört oder unbrauchbar gemacht werden können. FEIG übernimmt für derartiges Handeln unabhängig davon ob dies bewusst oder unbewusst erfolgte keinerlei Gewährleistung.
3. FEIG liefert Ihnen die Software "wie besehen" ohne jegliche Gewährleistung. FEIG kann für die Leistung oder die Ergebnisse, die Sie durch die Nutzung der Software erzielen, nicht garantieren. FEIG übernimmt keine Gewährleistung oder Garantie dafür, dass keine Schutzrechte Dritter verletzt werden, auch nicht dafür, dass die Software für irgendeinen bestimmten Zweck geeignet ist.
4. FEIG weist ausdrücklich darauf hin, dass das Lizenzmaterial nicht für den Einsatz mit oder in medizinischen Geräten oder für Geräte für lebenserhaltende Maßnahmen konzipiert ist, bei denen ein Fehler eine Gefahr für menschliches Leben oder für die gesundheitliche Unversehrtheit zur Folge haben kann.
   Der Anwender des Lizenzmaterials ist dafür verantwortlich, geeignete Maßnahmen zu ergreifen um Gefahren, Schäden oder Verletzungen zu vermeiden.

§4 Schlussbestimmungen

1. Dieser Vertrag enthält die vollständigen Lizenzbestimmungen und ersetzt alle eventuell vorangegangenen Regelungen und Absprachen. Änderungen und Ergänzungen bedürfen der Schriftform.
2. Sollte eine der in diesem Vertrag enthaltenen Bestimmungen unwirksam sein oder werden, so wird die Gültigkeit der übrigen Bestimmungen hierdurch nicht berührt. Beide Vertragsparteien verpflichten sich, die unwirksame Bestimmung durch eine solche wirksame Bestimmung zu ersetzen, die dem wirtschaftlichem Zweck der zu ersetzenden Bestimmung am nächsten kommt.
3. Dieser Vertrag unterliegt dem Recht der Bundesrepublik Deutschland. Gerichtsstand ist Frankfurt a. M.

Inhalt:

Anmerkungen zur Dokumentation dieser Bibliothek

Dieses Handbuch beschreibt eine Software-Bibliothek, die Ihnen auch als kommentierter Sourcecode vorliegt. Aus diesem Grund wird darauf verzichtet, mehr zu dokumentieren als unbedingt für das Verständnis der Funktionsweise und der Verwendung der Klassen notwendig ist. Es wird vorausgesetzt, dass der Anwender dieser Bibliothek den Sourcecode lesen und sich in die Details mit Hilfe dieser Dokumentation, der Header-Dateien und den eingefügten Kommentaren selbst zurechtfinden lernt.

Zum Verständnis der internen Programmabläufe müssen immer auch die Systemhandbücher der eingesetzten OBID®-Leser und der OBID®-Funktionsbibliotheken herangezogen werden.

FEIG ELECTRONIC GmbH verzichtet darauf, Informationen zu OBID®-Lesern in verschiedenen Handbüchern mehrfach darzustellen oder Querverweise auf bestimmte Seitenzahlen eines anderen Dokumentes einzubauen. Dies ist durch die ständige Aktualisierung der Handbücher notwendig und vermeidet Irrtümer durch Informationen in veralteten Dokumenten. Dem Anwender dieser Bibliothek sei deshalb empfohlen, sich ständig zu vergewissern, dass er die aktuellen Handbücher vorliegen hat. Diese kann er selbstverständlich jederzeit bei FEIG ELECTRONIC GmbH anfordern.

Wichtige Hinweise:

Sie dürfen diese Bibliothek nur verwenden, wenn Sie zuvor den umseitig abgedruckten Lizenzbestimmungen zugestimmt haben.

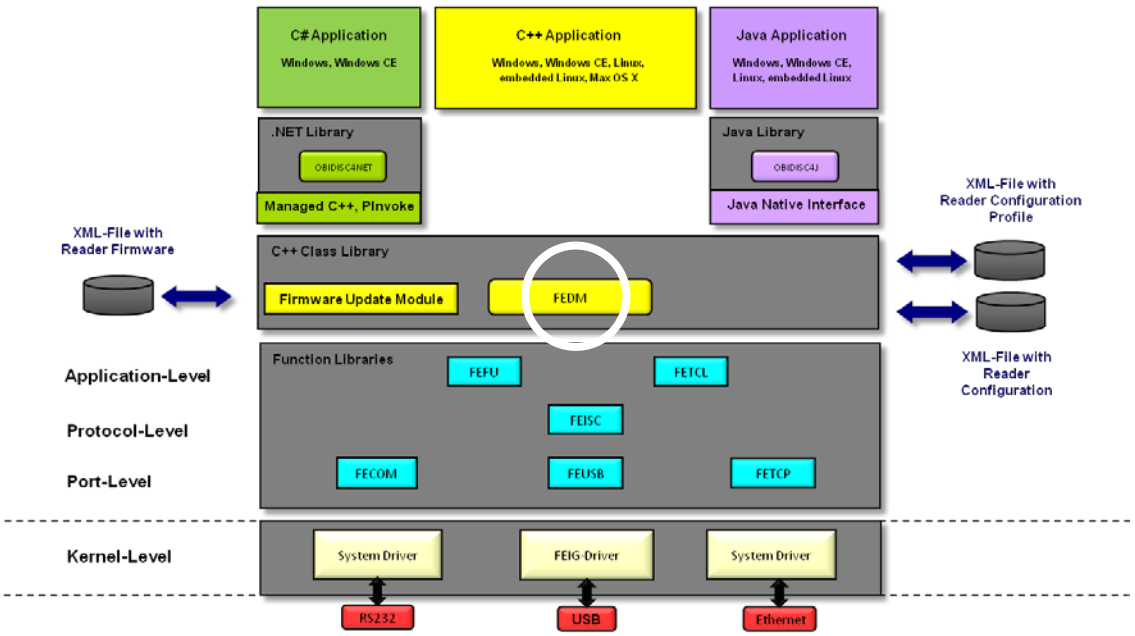
Sourcecode ist von jedermann veränderbar. Arbeiten Sie deshalb nur mit Bibliotheken, die Sie direkt von FEIG ELECTRONIC GmbH erhalten haben. Unabhängig davon ist die Weitergabe des Sourcecodes verboten.

1. Einführung

Mit der C++ Klassenbibliothek ID FEDM stellt Ihnen FEIG ELECTRONIC GmbH eine weitere Komponente zur Verfügung, mit der Ihnen die Entwicklung von Anwendungsprogrammen für OBID®-RFID-Leser vereinfacht werden soll.

Die C++ Klassenbibliothek ID FEDM unterstützt alle OBID®-Leserfamilien und kann als weitere Protokollschicht oberhalb der OBID®-Funktionsbibliotheken angesehen werden.

Die Bibliothek FEDM bildet für C++ die oberste Ebene in dem mehrschichtigen, hierarchisch strukturierten Aufbau von FEIG-Bibliotheken. Das nachfolgende Bild zeigt am Beispiel der OBID i-*scan*® und OBID® *classic-pro* Leserfamilien die Übersicht über alle Bibliotheken.



Mit der C++ Klassenbibliothek ID FEDM wird ein Organisationsprinzip für alle OBID®-Leserfamilien eingeführt, das es erlaubt, für alle OBID®-Leser ähnliche Programmstrukturen unabhängig der verwendeten Leser zu erstellen. Dieses Organisationsprinzip, das in Form ähnlicher Funktionsschnittstellen auch in den OBID®-Funktionsbibliotheken realisiert wurde, wird hier auf Datencontainer und die Steuerung des Datenflusses und des Protokollverkehrs ausgedehnt.

Trotz des einheitlichen Organisationsprinzips ist die Sicht auf die speicherbaren Daten der Leser und Transponder noch immer auf einer sehr niedrigen Ebene angesiedelt. Das bedeutet, dass man als Programmierer mit Leserparametern in Bits und Bytes konfrontiert wird und Transponderdaten nur als unorganisierte Datenmengen angeboten bekommt. Das hat einerseits den Vorteil, dass man auf alles einen Zugriff hat, andererseits aber mehrere Operationen hintereinander ausführen muss, wenn man z. B. nur einige Daten in einen Transponder schreiben möchte. Eine weitere Vereinfachung hinsichtlich der Abstraktion von Datenflüssen und Aktionen bleibt einer darüber liegenden Modulschicht vorbehalten.

Mit der C++ Klassenbibliothek ID FEDM wird eine einfache Art der Serialisierung von Daten der Leserkonfiguration angeboten. Damit ist es möglich, eine komplette Leserkonfiguration in einer XML-Datei abzuspeichern, später wieder zu laden und sie in den Leser zu transferieren.

Die Dokumentation zur C++ Klassenbibliothek ist zweigeteilt: Dieses Dokument beschreibt ausschließlich die Basisklassen und gemeinsamen Merkmale der spezialisierten Leserklassen. Die detaillierte Dokumentation der Leserklassen ist in separate Handbücher ausgegliedert.

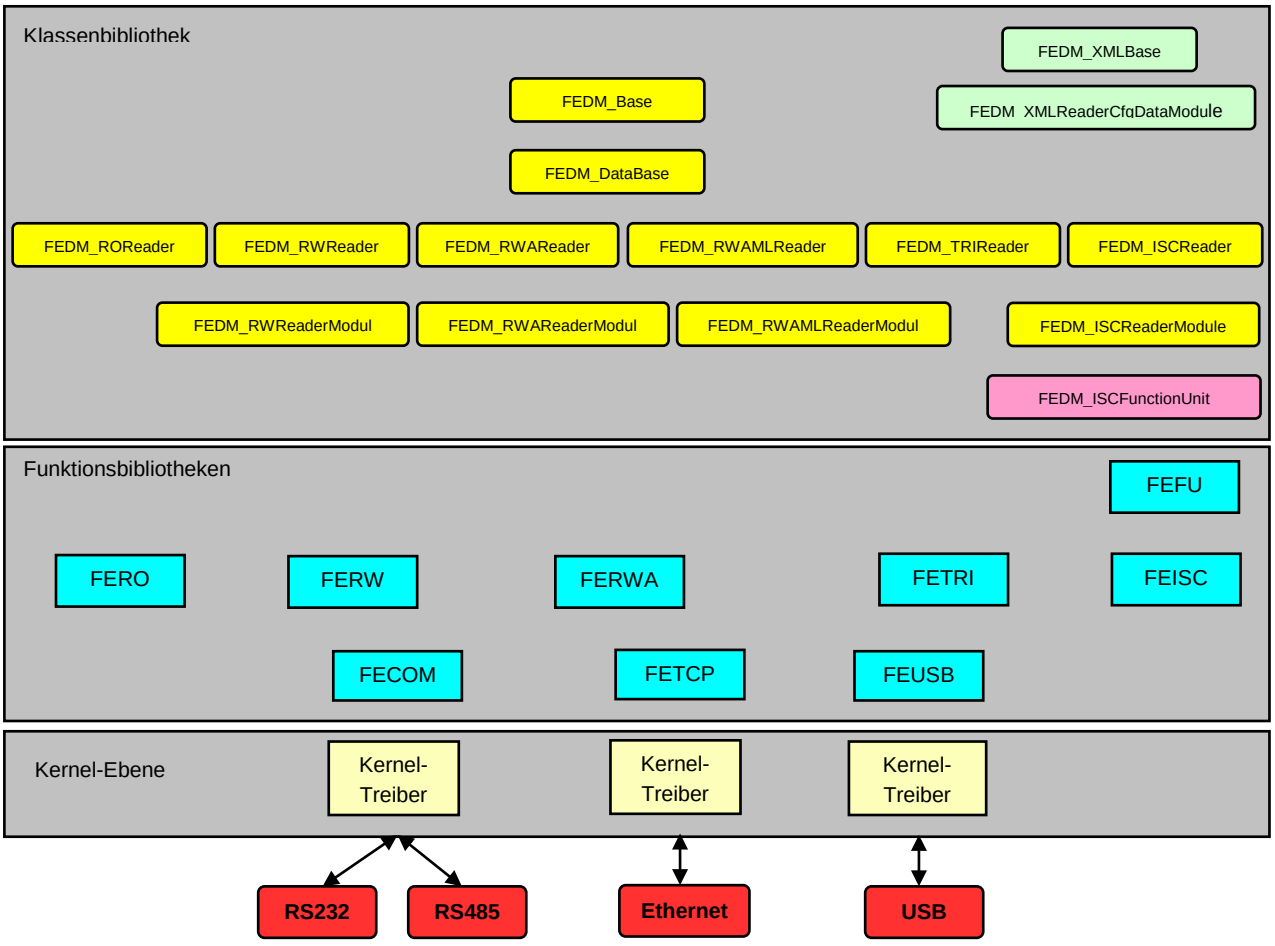
Wichtiger Hinweis:

Die Klassenbibliothek ID FEDM ist einem ständigen Anpassungsprozeß ausgesetzt. Wir werden uns bemühen, den dokumentierten Stand beizubehalten. Änderungen sind trotzdem nicht auszuschließen.

1.1. Übersicht über alle OBID-Komponenten

Die C++ Klassenbibliothek ID FEDM baut auf anderen OBID®-Komponenten auf. In der Übersicht ist dargestellt, welche anderen Komponenten für die jeweilige Leserfamilie und vorgesehene Schnittstelle notwendig ist. Die Funktionsbibliotheken sind alle in Form dynamisch linkfähiger Bibliotheken (DLL bzw. LIB) ausgeführt.

Informationen zu den OBID®-Funktionsbibliotheken sind in den jeweiligen Handbüchern zu finden. FEIG ELECTRONIC liefert Ihnen auf Anfrage gerne diese Dokumente nach, falls Ihnen eine fehlen sollte.



Die Schichtung von Bibliotheken, die in der Übersicht erkennbar ist, spiegelt den Grad der Spezialisierung wieder. Die unterste (physische) Schicht mit den Kernel-Treibern übernimmt den hardwarenahen Protokolltransfer. Die darüber liegende Protokolltransfer-Schicht mit FECOM, FETCP bzw. FEUSB.DLL bietet einer Applikation die erste Funktionsschnittstelle zu den physischen Schnittstellen. Eine weitere Schicht darüber befindet sich die spezialisierte Protokollschicht für OBID®-Leser (FERO, FERW, FERWA, FETRI, FEISC). Mit dieser Schicht ist bereits eine vereinfachte, aber vollständige Kommunikation mit OBID®-Lesern möglich.

In komplexeren Applikationen müssen Organisationsformen für die mit den Protokollen transferierten Daten aufgebaut werden. Dies ist die Aufgabe der C++Klassenbibliothek ID FEDM. Sie ist wiederum eine Schicht über der spezialisierten Protokollschicht für OBID®-Leser angesiedelt. Sie muss nun einheitliche Organisationsstrukturen für alle OBID®-Leserfamilien anbieten. Umgesetzt wird dieses Ziel mit den Basisklassen, den spezialisierten Leserklassen und dem Einsatz objekt-orientierter Methoden, wie überladenen Methoden, abstrakte Basisklasse (FEDM_DataBase) und der Verwendung der Standard Template Library (STL).

Die C++ Klassenbibliothek ID FEDM muss nicht in kompletter Form in eine Applikation integriert werden. Je nach verwendeter OBID®-Leserfamilie nutzt man nur die spezialisierte Leserklasse und die Basisklassen, sowie die globalen Funktionen und Konstanten. Die für die verwendete Leserklasse darunterliegenden Funktionsbibliotheken sind für die Funktionsweise der Leserklasse innerhalb der C++ Klassenbibliothek ID FEDM allerdings zwingend notwendig.

1.2. Unterstützte 32- und 64-Bit Betriebssysteme

Verwendet werden kann die Bibliothek mit folgenden Betriebssystemen:

Betriebssystem	Ausführung		Anmerkungen
	32-Bit	64-Bit	
Windows XP	X	(X)	bei 64-Bit nur mit 32-Bit Laufzeitsystem
Windows Vista / 7 / 8	X	X	
Windows CE	X	-	
Linux	X	X	
Android	X		Auf Anfrage
Apple Max OS X	-	X	ab V10.7.3, Architektur x86_64

1.3. Unterstützte Entwicklungsumgebungen

Betriebssystem	Entwicklungsumgebung	Unterstützung
Windows XP / Vista / 7 / 8	Visual Studio 6	auf Anfrage
	Visual Studio 2005 / 2008 / 2010 / 2012 / 2013	ja, ab Professional Version
	Borland C++ Builder	auf Anfrage
	Embarcadero C++ Builder	auf Anfrage
Windows CE	eMbedded Visual C++ 4	nein
	Visual Studio 2005 / 2008	ja, ab Professional Version
Linux	GCC	ja, für 32-Bit Projekte
Mac OS X	GCC	ja, für Projekte mit x86_64 Architektur
	Xcode ≥ V4.3.2	ja, für Projekte mit x86_64 Architektur

2. Änderungen gegenüber der Vorversion

- Keine Änderungen in der Basisfunktionalität

Bitte beachten Sie auch die Änderungshistorie im Anhang.

3. Installation

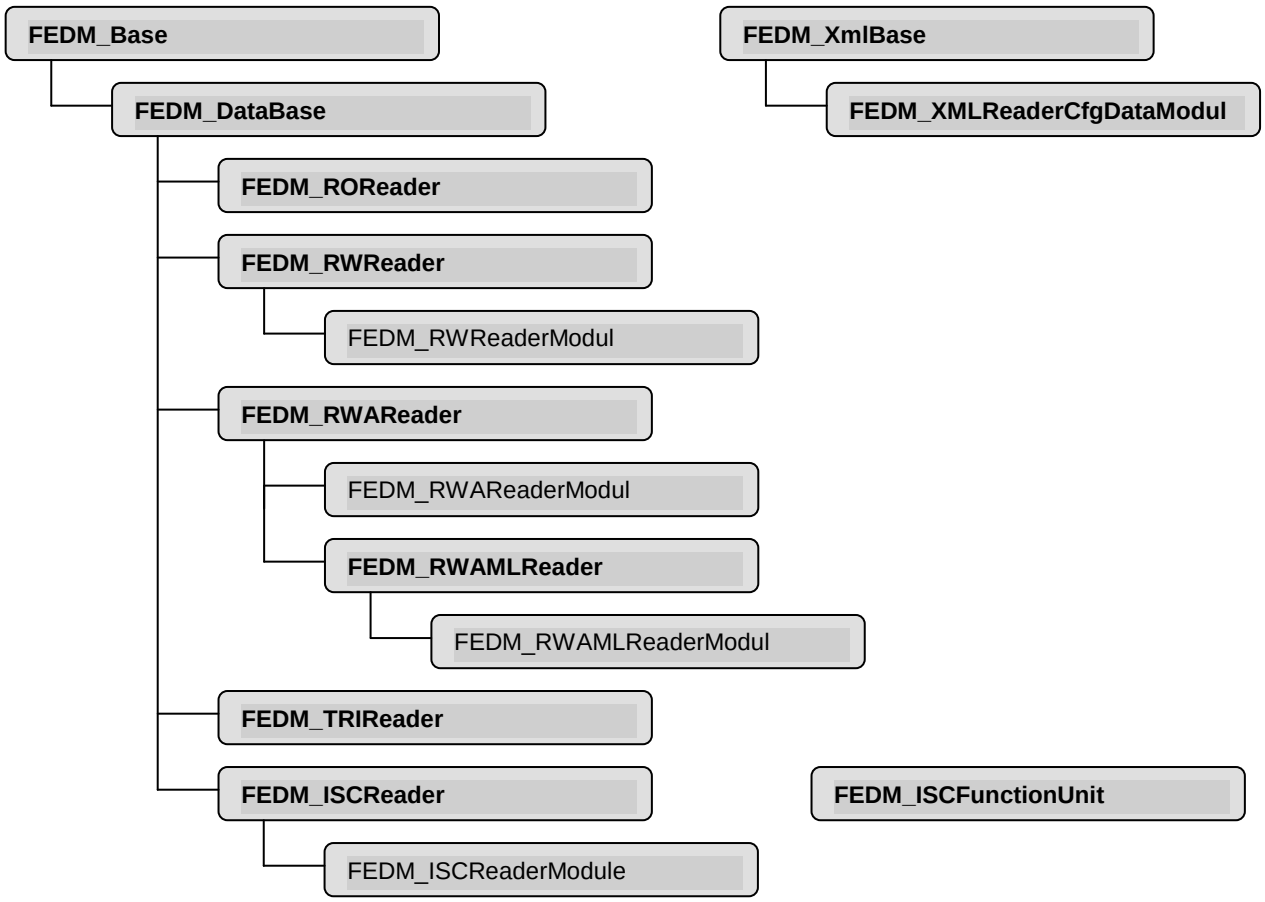
Die Installation wird im Teil B (H10202-xd-ID-B) der Dokumentation zur Klassenbibliothek beschrieben.

4. Übersicht

Je nach Anforderung, die sich dem Anwender stellt, benötigt er eine Grundklasse, auf der er seine spezielle Leserklasse aufbauen möchte, oder er nutzt die vorhandene Leserklasse als Komponente mit definierter Schnittstelle – quasi als Black-Box-Objekt. Für beide Anwendungsfälle ist nachfolgend zur schnellen Übersicht je ein Diagramm dargestellt.

4.1. Klassenbaum

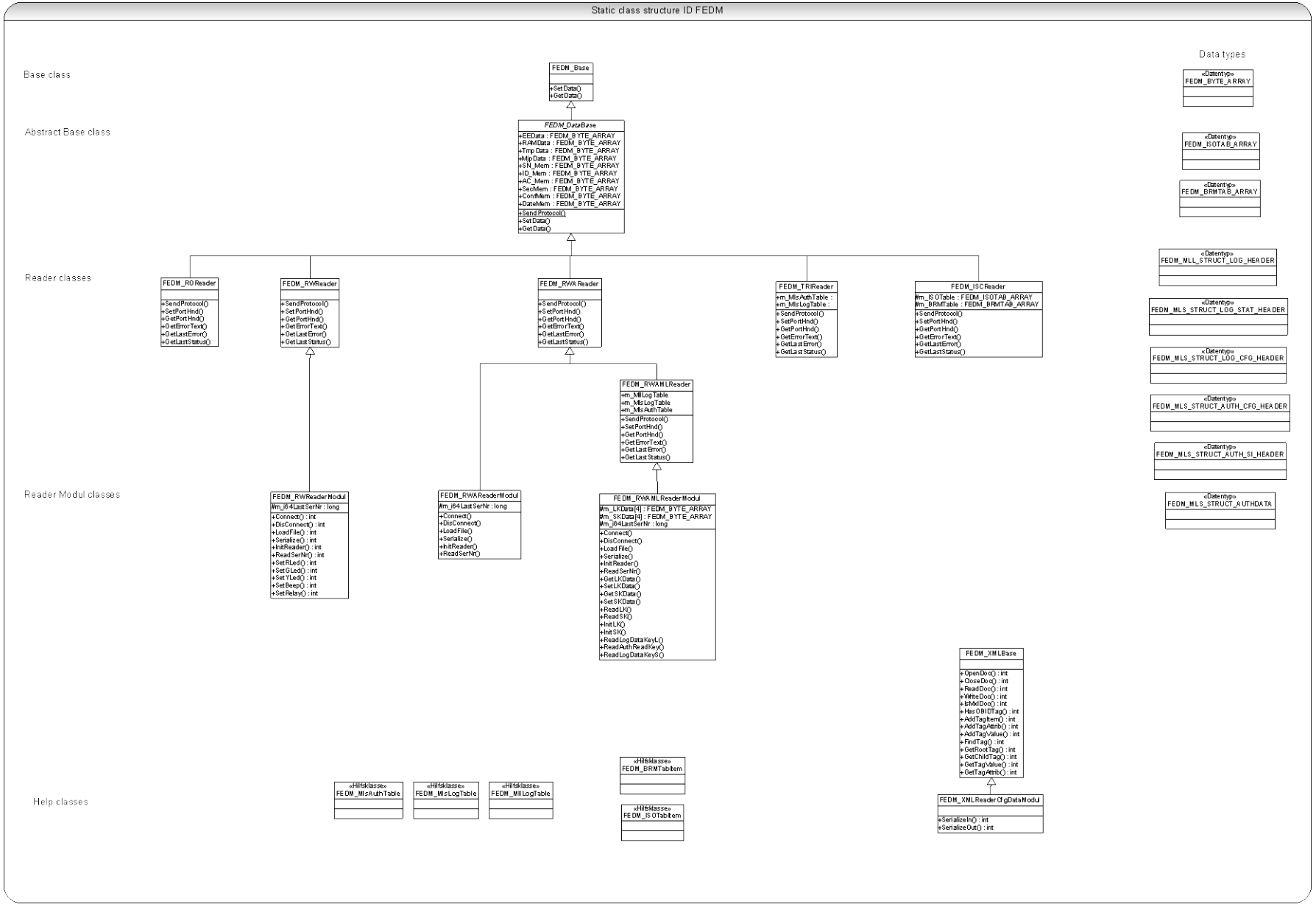
Nachfolgend ist der Klassenbaum für eine Schnellübersicht dargestellt. Die Klassen FEDM_RWReaderModul, FEDM_RWAReaderModul und FEDM_RWAMLReaderModul sind z. Zt. im Aufbau.



4.2. Klassenstruktur-Diagramm

Das Struktur-Diagramm (s. nächste Seite) zeigt die statische Struktur der C++ Klassenbibliothek. Klassenbaum, FEDM-spezifische Datentypen und Hilfsklassen sind dargestellt.

Die Klassen im Diagramm enthalten nur einige wichtige Attribute und Methoden.

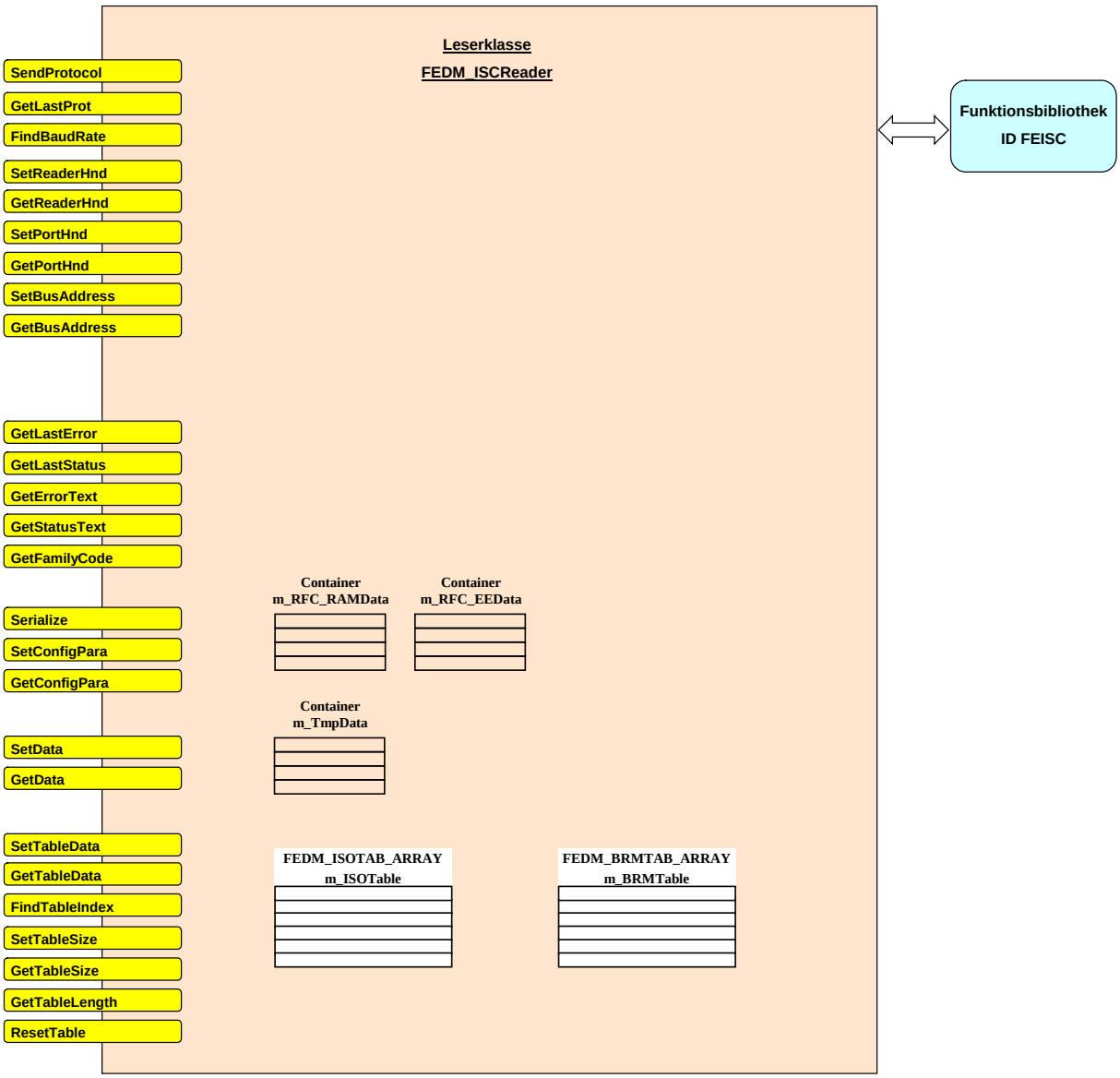

Static class structure ID FEDM
Base class
Abstract Base class
Reader classes
Reader Modul classes
Help classes
Data types
FEDM_Base
+Set Data()
+Get Data()
FEDM_DataBase
+EEData : FEDM_BYTE_ARRAY
+RAMData : FEDM_BYTE_ARRAY
+Tmp Data : FEDM_BYTE_ARRAY
+MipData : FEDM_BYTE_ARRAY
+SN_Mem : FEDM_BYTE_ARRAY
+ID_Mem : FEDM_BYTE_ARRAY
+AC_Mem : FEDM_BYTE_ARRAY
+SecMem : FEDM_BYTE_ARRAY
+ConfMem : FEDM_BYTE_ARRAY
+DateMem : FEDM_BYTE_ARRAY
+Send Protocol()
+Set Data()
+Get Data()
FEDM_RO Reader
+Send Protocol()
+Set Port Hnd()
+Get Port Hnd()
+Get Error Text()
+Get Last Error()
+Get Last Status()
FEDM_RW Reader
+Send Protocol()
+Set Port Hnd()
+Get Port Hnd()
+Get Error Text()
+Get Last Error()
+Get Last Status()
FEDM_RWA Reader
+Send Protocol()
+Set Port Hnd()
+Get Port Hnd()
+Get Error Text()
+Get Last Error()
+Get Last Status()
FEDM_TRI Reader
+m_MlsAuthTable :
+m_MlsLogTable :
+Send Protocol()
+Set Port Hnd()
+Get Port Hnd()
+Get Error Text()
+Get Last Error()
+Get Last Status()
FEDM_ISC Reader
#m_ISOTable : FEDM_ISOTAB_ARRAY
#m_BRMTable : FEDM_BRMTAB_ARRAY
+Send Protocol()
+Set Port Hnd()
+Get Port Hnd()
+Get Error Text()
+Get Last Error()
+Get Last Status()
FEDM_RWAMLReader
+m_MlLog Table
+m_MlsLogTable
+m_MlsAuthTable
+Send Protocol()
+Set Port Hnd()
+Get Port Hnd()
+Get Error Text()
+Get Last Error()
+Get Last Status()
FEDM_RW Reader Modul
#m_i64 Last Ser Nr : long
+Connect() : int
+DisConnect() : int
+Load File() : int
+Serialize() : int
+Init Reader() : int
+Read Ser Nr() : int
+Set RLed() : int
+Set GLed() : int
+Set YLed() : int
+Set Beep() : int
+Set Relay() : int
FEDM_RWA Reader Modul
#m_i64 Last Ser Nr : long
+Connect()
+DisConnect()
+Load File()
+Serialize()
+Init Reader()
+Read Ser Nr()
FEDM_RWAML Reader Modul
#m_LKData[4] : FEDM_BYTE_ARRAY
#m_SKData[4] : FEDM_BYTE_ARRAY
#m_i64 Last Ser Nr : long
+Connect()
+DisConnect()
+Load File()
+Serialize()
+Init Reader()
+Read Ser Nr()
+Get LKData()
+Set LKData()
+Get SKData()
+Set SKData()
+Read LK()
+Read SK()
+Init LK()
+Init SK()
+Read Log Data Key LK()
+Read Auth Read Key()
+Read Log Data Key SK()
FEDM_XMLBase
+Open Doc() : int
+Close Doc() : int
+Read Doc() : int
+Write Doc() : int
+Is Mid Doc() : int
+Has OBID Tag() : int
+Add Tag Item() : int
+Add Tag Attrib() : int
+Add Tag Value() : int
+Find Tag() : int
+Get Root Tag() : int
+Get Child Tag() : int
+Get Tag Value() : int
+Get Tag Attrib() : int
FEDM_XML Reader Cfg Data Modul
+Serialize In() : int
+Serialize Out() : int
«Hilfsklasse» FEDM_MlsAuthTable
«Hilfsklasse» FEDM_MlsLogTable
«Hilfsklasse» FEDM_MlLogTable
«Hilfsklasse» FEDM_BRMTabItem
«Hilfsklasse» FEDM_ISOTabItem
«Datentyp» FEDM_BYTE_ARRAY
«Datentyp» FEDM_ISOTAB_ARRAY
«Datentyp» FEDM_BRMTAB_ARRAY
«Datentyp» FEDM_MLL_STRUCT_LOG_HEADER
«Datentyp» FEDM_MLS_STRUCT_LOG_STAT_HEADER
«Datentyp» FEDM_MLS_STRUCT_LOG_CFG_HEADER
«Datentyp» FEDM_MLS_STRUCT_AUTH_CFG_HEADER
«Datentyp» FEDM_MLS_STRUCT_AUTH_SI_HEADER
«Datentyp» FEDM_MLS_STRUCT_AUTHDATA

4.3. Komponenten-Diagramm

Das Komponenten-Diagramm zeigt eine andere Sicht auf die C++ Klassenbibliothek ID FEDM.

Es sind nur die wichtigsten Methoden dargestellt. Auf Attribute wurde hier verzichtet. Ein genaues und vollständiges Komponentendiagramm ist in der Dokumentation der jeweiligen Leserklasse enthalten.

Die meisten Membermethoden bilden eine für alle Leserklassen einheitliche, typ-unabhängige Schnittstelle. Damit ist es möglich, verschiedene Leserfamilien in einer Anwendung mit denselben Algorithmen zu betreiben.



4.4. Threadsicherheit

Alle FEIG-Bibliotheken sind prinzipiell nicht vollständig Threadsicher. Unter Beachtung einiger Regeln kann man dennoch Parallelität in der Ausführung von Kommunikationsaufgaben und damit praktische Threadsicherheit erreichen. Man muss auch wissen, dass alle OBID® RFID-Leser immer nur eine Aktion ausführen können, also synchron arbeiten.

Auf der Ebene der Transportschicht (FECOM, FEUSB, FETCP) kann über jede Verbindung nur synchron kommuniziert werden, weil auch die Leser nur synchron arbeiten. Threadsicher sind die Port-Objekte untereinander, weil diese unabhängig voneinander sind. Es ist demnach möglich, dass z. B. zwei Threads mit zwei RFID-Lesern über zwei verschiedene Serielle Schnittstellen kommunizieren. Es ist aber nicht möglich, dass zwei Threads über eine Serielle Schnittstelle vom Typ RS485 oder RS422 mit zwei Lesern kommunizieren, da beide Leser an derselben Busleitung betrieben werden. Eine weitere Einschränkung gilt für die Scan-Funktionen in der FEUSB, die generell nicht Threadsicher sein können, da eine globale Aktion über den gesamten USB gestartet wird. Parallelität beim Öffnen und Schließen von Verbindungen für die Serielle Schnittstelle und USB sollte in der Applikation serialisiert werden, damit es nicht zu gegenseitigen Beeinflussung kommt.

Auf der Ebene der Protokollschicht ist Parallelität über separate Leser-Objekte möglich, wenn jedes Leser-Objekt mit einer eigenen Kommunikations-Schnittstelle verbunden ist. Eine Ausnahme gilt für die vier speziellen Funktionen FEISC_BuildxxProtocol, FEISC_SplitxxProtocol, die eine globale Variable für Protokolldaten nutzen.

Die Bibliothek FEFU für Externe Funktionseinheiten bietet keine Threadsicherheit. Über sie kann immer nur ein Thread kommunizieren und die Threadsicherheit muss in der Applikation hergestellt werden.

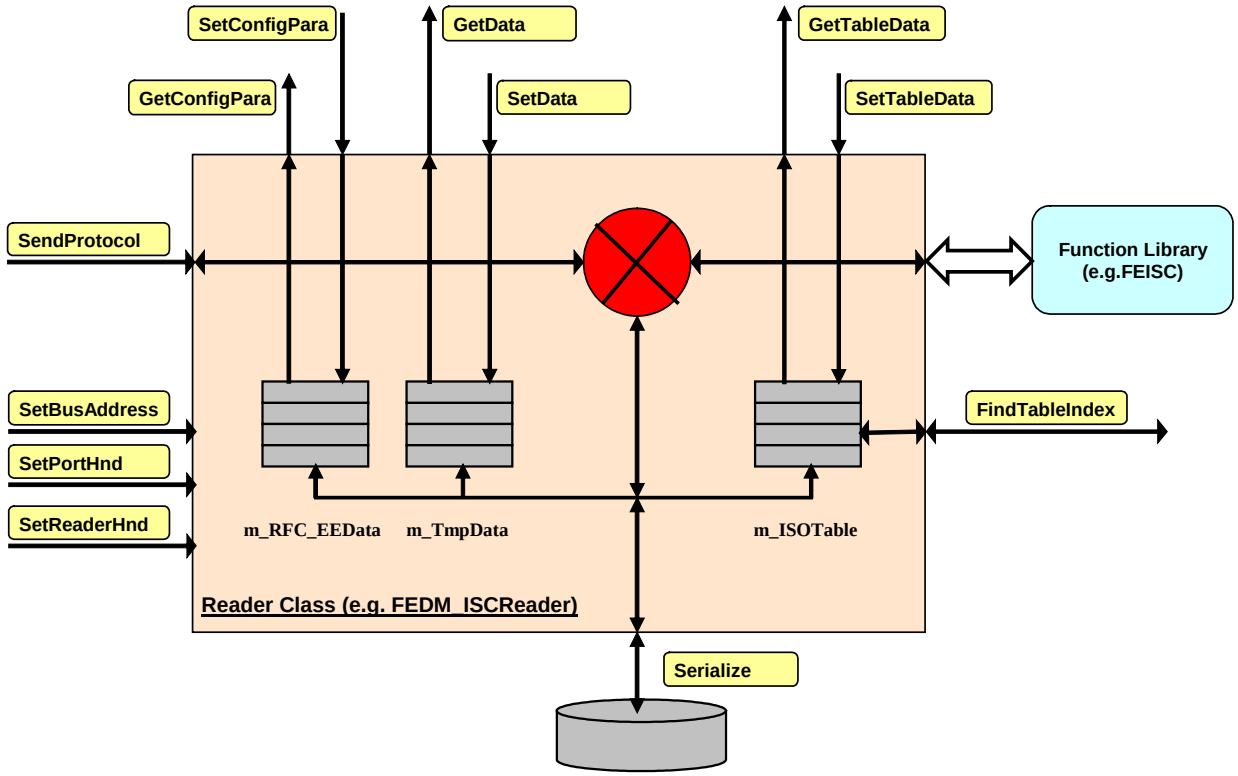
Die Bibliothek FETCL für ISO 14443-4 kompatible Transponder bietet Threadsicherheit, wenn jedes Transponder-Objekt mit einem anderen Leser verbunden ist, also nur eine APDU pro Leser gleichzeitig gesendet wird. Dass die Funktion FETCL_Apdu asynchron aufrufbar ist, bedeutet nicht, dass mehrere Aufrufe von FETCL_Apdu an dasselbe Transponder-Objekt möglich sind, denn APDUs werden intern nicht in einem Stack gespeichert.

Auf der Ebene der Klassenbibliotheken (FEDM, OBIDISC4J, OBIDISC4NET) ist Parallelität möglich, wenn pro Leser-Objekt nur eine Aktion ausgeführt wird. Threadsicherheit für jedes Leser-Objekt muss also in der Applikation hergestellt werden. Parallelität beim Öffnen und Schließen von Verbindungen für die Serielle Schnittstelle und USB (ConnectCOMM, ConnectUSB) sollte in der Applikation vermieden werden, damit es nicht zu gegenseitigen Beeinflussung kommt. Bei Verwendung der FunctionUnit-Objekte für Externe Funktionseinheiten gilt, dass global immer nur über ein FunctionUnit-Objekt kommuniziert werden darf, auch wenn diese an unterschiedlichen Lesern angeschlossen sind. Dies liegt an der darunter liegenden Bibliothek FEFU, die keine Threadsicherheit bietet.

5. Grundlegende Eigenschaften der Klassenbibliothek

5.1. Interner Aufbau

Die Funktion einer Leserklasse – auf die man letztlich beim Einsatz der Klassenbibliothek die Sicht reduzieren kann – läßt sich sehr schön am nachfolgenden Schaubild verdeutlichen: In der Vertikalen aufgetragen sind die Datenströme, die mit den (überladenen) Methoden GetData bzw. GetConfigPara und SetData bzw. SetConfigPara, sowie GetTableData[1] und SetTableData bewegt werden. Zusätzlich ist mit der Methode Serialize der Datenfluß zwischen einer Instanz der Leserklasse und einer Datei möglich.



In der Horizontalen sieht man den Steuerfluss, der mit der Methode SendProtocol, der einzigen Kommunikationsmethode, ausgelöst wird. Sie holt sich intern selbständig vor der Ausgabe des Sendeprotokolls alle notwendigen Daten aus den integrierten Datencontainern und legt auch dort die empfangenen Protokolldaten wieder ab. Somit muss das Anwendungsprogramm vor dem Aufruf von SendProtocol alle für dieses Protokoll notwendigen Daten in die entsprechenden Datencontainer an die richtigen Stellen schreiben. Ebenso sind die Empfangsdaten an bestimmten Stellen in entsprechenden Datencontainern hinterlegt.

[1] Nicht alle Leserklassen haben eine Implementierung von GetTableData und SetTableData, weil sie keine Tabelle integriert haben.

Der Schlüssel zu den Protokolldaten sind sogenannte Zugriffskonstanten für temporäre Protokolldaten (z.B. `FEDM_ISC_TMP_READER_INFO_MODE`) und Namespaces für Parameter der Leserkonfiguration (z.B. `ReaderConfig::OperatingMode::Mode`). Für jede Leserklasse können einige Dutzend bis hundert Konstanten und Namespaces definiert sein. Ihr Aufbau ist für alle Leserklassen gleich und von besonderer Bedeutung. Er wird genau in 5.3. Zugriffskonstanten für temporäre Protokolldaten und 5.4. Namespaces für Konfigurationsparameter erläutert. Da die Zugriffskonstanten von elementarer Bedeutung für den gesamten Protokolltransfer einer jeden Leserklasse sind, werden sie in den Dokumentationen zu den Leserklassen genau in ihrem Zusammenhang beschrieben. Die Abbildung eines jeden Konfigurationsparameters in einen Parameter-Namespace ist in den Systemhandbüchern der Leserfamilien dokumentiert.

5.2. Datencontainer

Datencontainer haben die Aufgabe, alle Parameter bzw. Transponderdaten strukturiert zu verwalten. Intern sind alle Datencontainer als Byte-Arrays im Motorola-Format (Big Endian) organisiert. Dieses Format entspricht jedem OBID®-Leser. Die Umwandlung in das für Intel-basierte PC's notwendige Intel-Format (Little Endian) übernehmen die überladenen Zugriffsmethoden.

Die Byte-Arrays sind organisiert in Blöcke zu je 32Byte, 16 Byte oder 4 Byte. Auch diese Organisation entspricht dem der Leser bzw. Transponder.

Insgesamt 11 Datencontainer sind in der abstrakten Basisklasse FEDM_DataBase integriert. Sie haben alle die Größe 0. Die Größe der benötigten Datencontainer wird von der abgeleiteten Leserklasse festlegt. Ungenutzte Datencontainer behalten die Größe 0.

Datencontainer	Beschreibung
m_RFC_EEData	für Konfigurationsparameter des RF-Controllers im Lesers
m_RFC_RAMData	für temp. Konfigurationsparameter des RF-Controllers im Lesers
m_TmpData	für allgemeine temp. Protokolldaten

Nachfolgende Datencontainer sind speziell für classic Leserfamilie (RW / RWA)

Datencontainer	Beschreibung
m_MjpData	für temp. Protokolldaten eines multijob-Polls
m_SN_Mem	für Seriennummer des Transponders
m_ID_Mem	für ID-Nummern des Transponders
m_AC_Mem	für Accountdaten des Transponders
m_PubMem	für public Datenblöcke des Transponders
m_SecMem	für secret Datenblöcke des Transponders

Datencontainer	Beschreibung
m_ConfMem	für Konfigurationsdaten des Transponders
m_DateMem	für Datumsdaten des Transponders

5.2.1. Datenaustausch

Der Zugriff auf die temporären Protokolldaten ist vorrangig mit den überladenen Methoden SetData und GetData möglich. Mit jedem Methodenaufruf kann genau ein Protokoll-Parameter gelesen bzw. geschrieben werden, der durch eine Zugriffskonstante (s. 5.3. Zugriffskonstanten für temporäre Protokolldaten) identifiziert wird.

Der Datenaustausch mit Konfigurationsparametern wird mit den überladenen Methoden SetConfigPara und GetConfigPara realisiert.

Alternativ kann man direkt auf die Bytes der Datencontainer zugreifen, weil sie in der Klasse FEDM_DataBase public angelegt sind. Diese Methode sollte aber nur für den Byte-Zugriff verwendet werden.

Die nachfolgenden Kapitel zeigen die Verwendung von GetData und SetData für verschiedene Datentypen. GetConfigPara und SetConfigPara sind analog zu verwenden, mit dem Unterschied, dass für die Zugriffskonstante der String des Namespaces übergeben wird.

5.2.1.1. Konstante Daten

```
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. Datentyp 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. Datentyp 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. Datentyp 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. Datentyp 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. Datentyp int

int als Datentyp kann nicht direkt unterstützt werden, weil der Datentyp BOOL bereits als int definiert ist. Stattdessen muss der cast-Operator UINT bzw. unsigned int vorangestellt werden.

```
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. Datentyp 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. Datentyp __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. Datentyp CString bzw. AnsiString (VC++, C++Builder) und STL-string

ALLE Daten, die mit einer String-Methode (CString, AnsiString, string) gelesen werden, sind "Hex"-Strings. Das bedeutet z. B. , dass der numerische Wert 159 nicht als "159" sondern als "9F" übergeben wird. FEDM-kompatible String-Werte bestehen somit immer aus einer geraden Anzahl Zeichen. Zur Konvertierung von String-Werten in andere Datentypen bzw. umgekehrt steht die Funktionssammlung in FEDM_Functions (s. Anhang 7.1. FEDM_Functions) zur Verfügung.

Zur Umwandlung von numerischen Werten in Strings, die aus dem o. g. Beispiel ein "159" machen, bedient man sich am Besten den Funktionen der ANSI C-Library (z. B. sprintf und zur Rückumwandlung 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. Direkter, adressierter Zugriff

In einigen Programmier-Fällen (z.B. Schleifen, Kopieroperationen) sind die Zugriffsparameter zu statisch. Zur Lösung dieser Aufgaben steht eine zweite Sammlung parametrisierter GetData- und SetData-Methoden zur Verfügung. Unterstützt werden diese Methoden durch einige nützliche Hilfsfunktionen in FEDM_Functions (s. Anhang 7.1. FEDM_Functions).

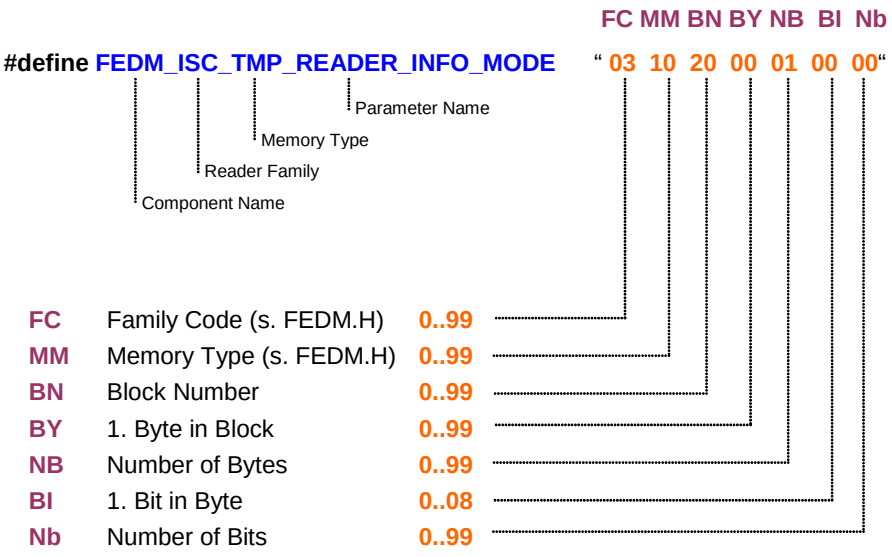
Als Datentypen werden UCHAR, UCHAR[], UINT, __int64 und CString bzw. AnsiString unterstützt. Die Verwendung ist analog zu den oben dargestellten Beispielen.

```
CHAR ucVersionInfo[11];
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. Zugriffskonstanten für temporäre Protokolldaten

Eine zentrale Rolle im Datenverkehr zwischen Anwendungsprogramm und dem Datencontainer für temporäre Protokolldaten der Klassenbibliothek, sowie innerhalb der Klassenbibliothek zwischen Protokollmethode und Datencontainer spielen die Zugriffskonstanten. Sie identifizieren einerseits den Protokoll-Parameter und enthalten gleichzeitig kodiert den Ablageort in einem der Datencontainer.

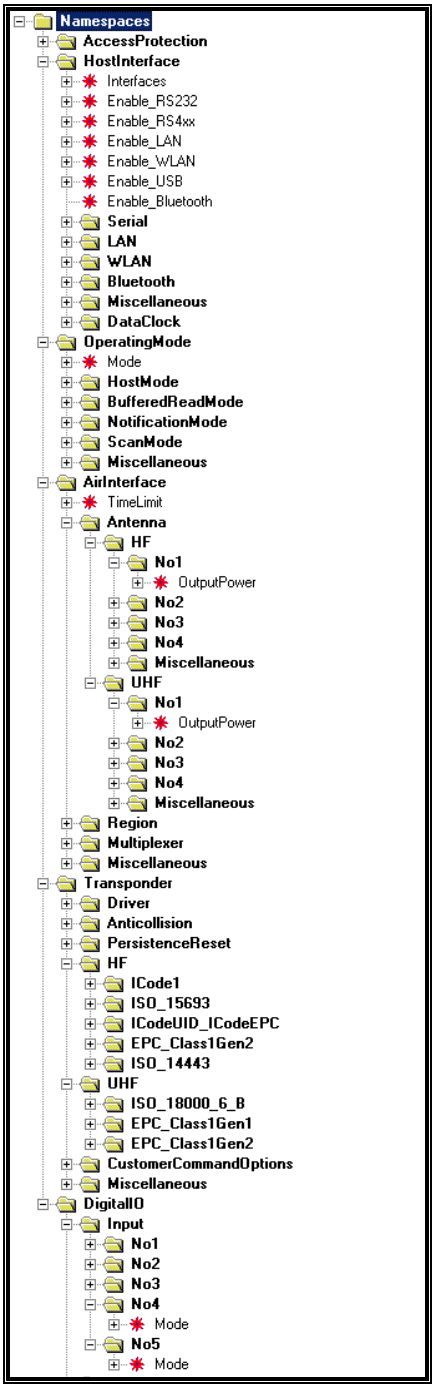
Eine Zugriffskonstante ist ein String und hat generell folgenden Aufbau:



Diese Zugriffskonstanten werden ausschließlich mit den Methoden SetData und GetData verwendet. Die Zugriffskonstante sagt nichts über den Datentyp eines Protokoll-Parameters aus. Dieser wird nur durch den Datentyp der Zugriffsmethode bestimmt. Man kann also die im obigen Beispiel dargestellte ReaderInfoMode entweder z. B. als Integer oder als String oder einen anderen plausiblen Datentyp mit SetData setzen (s. 5.2.1. Datenaustausch).

5.4. Namespaces für Konfigurationsparameter des Lesers

Der Datenverkehr zwischen dem Anwendungsprogramm und den Datencontainern für Konfigurationsparameter in der Klassenbibliothek wird mit überladenen Methoden realisiert, die als Identifikator einen Namen als String aus einem Namespace übergeben bekommen. Die Namespaces aller Konfigurationsparameter sind für alle Lesertypen vereinheitlicht worden, was den Vorteil der Vereinfachung der Programmstrukturierung zur Folge hat. Die Namespaces sind hierarchisch gegliedert in Gruppen und Untergruppen, die durch Doppelpunkte geteilt sind. Alle Namespaces zusammen ergeben eine Baumstruktur.



Ausschnitt aus der Baumstruktur der Namespaces

5.5. Tabellen

Einige Leserfamilien unterstützen Protokolle, die für mehrere Transponder Daten transportieren können (z. B. ISO-Commands der OBID i-*scan*®-Leserfamilie) oder einige aufeinanderfolgende Aktionen mit mehreren Transpondern einer anderen Leserfamilie ergeben einen Datenbestand (z.B. Daten von LogDataKeys der megalock Produktreihe), die eine Speicherung in den Containern unmöglich machen. Idealerweise speichert man diese Daten strukturiert in einer Tabelle. Obwohl diese Tabellen nicht in den Basisklassen enthalten sind, unterstützen die Basisklassen dennoch mit den abstrakten Methoden GetTableData, SetTableData und FindTableIndex den Zugriff auf und mit den Methoden GetTableSize, SetTableSize, GetTableLength und ResetTable die Verwaltung von in Leserklassen implementierten Tabellen. Zusätzlich kann mit der Methode VerifyTableData ein Vergleich der gesendeten zu den empfangenen Transponderdaten (z. Zt. nur für OBID i-*scan*® Familie) vorgenommen werden. Die Verwendung dieser Methoden ist für alle Leserfamilien somit vereinheitlicht.

Der Zugriff auf Daten von einer in der Leserklasse enthaltenen Tabelle mittels der Methoden SetTableData und GetTableData erfolgt zwar auch mit eindeutigen Konstanten wie in SetData und GetData. Sie stellen aber keinen String dar und enthalten somit auch keine Ortskodierung.

Stattdessen ist mit der Konstanten für den Tabellentyp, dem Tabellenindex und der Konstanten für den Tabellenwert eine eindeutige Identifizierung eines Tabellenwertes möglich.

Beispiel:

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

Als Datentypen werden bool, UCHAR, UINT, __int64, CString bzw. AnsiString und STL-string unterstützt.

Zwei weitere Set/GetTableData Methoden sind für den Blockzugriff auf Transponderdaten vorgesehen und unterstützen die Datentypen UCHAR[], CString bzw. AnsiString und STL-string.

5.6. Protokollverkehr

Der Protokollverkehr wird mit der einzigen Methode SendProtocol ausgelöst. Ihr übergibt man nur das Steuerbyte des gewünschten Protokolls. Alle für den Protokolltransfer notwendigen Daten werden den Datencontainern bzw. Tabellen entnommen. Es muss also sichergestellt sein, dass alle Protokolldaten vorher aktualisiert werden. SendProtocol ist als abstrakte Methode in FEDM_DataBase enthalten und wird erst in der Leserklasse implementiert.

Die meisten Leserfamilien sind busfähig und benötigen im Protokoll die Busadresse. Diese sollte mit der Methode SetBusAddress der Klasse FEDM_DataBase gesetzt werden.

Die Implementierung von SendProtokoll ruft Bibliotheksfunktionen in der darunter liegenden spezialisierten Protokollschicht für OBID®-Leser auf, die als ersten Parameter immer einen Reader-Handle verlangen. Diesen Reader-Handle muss man vor dem ersten Aufruf von SendProtocol mit der Membermethode SetReaderHnd initialiseren.

5.7. Initialisierungsmethoden

Vor der ersten Verwendung der Protokollmethode müssen einige Initialisierungen durchgeführt werden:

1. Busadresse — Die Busadresse des Lesers ist in der Klasse mit 255 voreingestellt. Eine andere Adresse stellt man mit der Methode SetBusAddress ein.
2. ReaderHandle — Der Handle eines Leserobjektes in einer FExxx-Funktionsbibliothek muss immer mit der Methode SetReaderHnd in einer Instanz der Leserklasse hinterlegt werden.
3. PortHandle — Der Handle einer Schnittstelle, die mit FECOM, FETCP bzw. FEUSB geöffnet wurde, muss im Leserobjekt einer FExxx-Funktionsbibliothek hinterlegt werden. Dies kann man entweder bei der Erzeugung des Leserobjektes mit FExxx_NewReader tun oder nachträglich mit der Methode SetPortHnd. Die nachträgliche Änderung ist auch immer dann möglich, wenn zur Laufzeit die Schnittstelle gewechselt werden soll.
4. Sprachunterstützung — Fehlertexte sind in mehreren Sprachen abrufbar. Die Sprache kann man mit der Methode SetLanguage einstellen. Voreingestellt ist Englisch.
5. Größe einer Tabelle — Manche Leserklassen stellen die Größe einer internen Tabelle nicht automatisch ein. Zur Einstellung nutzt man die Methode SetTableSize.
6. andere — Manche Leserklassen haben weitere Initialisierungsmethoden. Sie sind in den betreffenden Dokumenten zur Leserklasse beschrieben.

5.8. Serialisierung

Die beiden integrierten Serialisierungsmethoden Serialize erlauben das Speichern von Daten aus den Datencontainern in eine Datei bzw. laden Daten aus einer Datei in Datencontainer:

```
int Serialize(bool bRead, char* sFleName);
int Serialize(CArchive& ar, int iMemType);
```

Die erste und wichtigste Version von Serialize unterstützt das Datenformat XML. Mit einem Aufruf erfolgt die gesamte Serialisierung der Leserkonfiguration. Kenntnisse zu den Klassen FEDM_XMLBase und FEDM_XMLReaderCfgDataModul sind nicht notwendig.

Die zweite Version ist ausschließlich für MFC-basierte Applikationen geeignet. Mit jedem Aufruf von Serialize kann immer nur der Inhalt eines Datencontainers serialisiert werden. Kenntnisse zur MFC-Klasse CArchive werden vorrausgesetzt. Diese Methode ist für die OBID i-*scan*® Familie nicht mehr verfügbar.

5.9. Fehlerbehandlung

Fast alle Methoden der Klassenbibliothek führen intern Fehleruntersuchungen durch und geben im Fehlerfall einen negativen Wert zurück. Die Fehlercodes der Klassenbibliothek ID FEDM und der OBID®-Funktionsbibliotheken sind so in Bereiche organisiert, dass sie sich nicht überlappen können. Folgende Bereiche sind für die Klassenbibliothek ID FEDM und die OBID®-Funktionsbibliotheken reserviert:

Bibliothek	Wertebereich für Fehlercodes	Referenz
ID FEDM	-101 ... -999	8.2. Liste der Fehlercodes
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 FERW	-2000…-2099	H80591-xx-ID-B
ID FERO	-2100…-2199	H10201-xx-ID-B
ID FERWA	-3000…-3099	H80891-xx-ID-B
ID FETRI	-3100…-3199	H00903-xx-ID-C
ID FEISC	-4000…-4099	H9391-xx-ID-B
ID FEFU	-4100…-4199	H30801-xx-ID-B
IF FETCL	-4200…-4299	H50401-xx-ID-B

Mit der Methode GetErrorText der Leserklasse kann man zum Fehlercode einen Fehlertext abrufen. Der übergebene Fehlercode kann auch aus dem Bereich einer OBID®-Funktionsbibliothek kommen.

Der letzte Fehlercode wird im Datencontainer m_TmpData gespeichert und kann mit der Methode GetLastError abgerufen werden.

5.10. Sprachenunterstützung

Die Ausgabe von Fehlertexten ist in mehreren Sprachen möglich. Die Einstellung erfolgt mit der Methode SetLanguage der Basisklasse. Folgende Sprachen werden z. Zt. unterstützt:

Sprache	Übergabeparameter
Deutsch	7
Englisch (voreingestellt)	9

Wird ein Fehlertext aus einer OBID®-Funktionsbibliothek abgerufen, wird aus dieser der Text in der eingestellten Sprache übergeben.

Die Fehlertexte eignen sich normalerweise nicht für Anwendungsprogramme. Dafür sind sie zu speziell. Sie sind aber hilfreich und aussagekräftig in einer Debug-Version.

6. Klassenbeschreibung

6.1. FEDM_Base

Die Basisklasse FEDM_Base bildet die Basisklasse der C++ Klassenbibliothek ID FEDM. Sie enthält ausschließlich Methoden für die abgeleiteten Klassen. Mit einer Instanz dieser Klasse ist eine Leserkommunikation nicht möglich.

In der Headerdatei FEDM_Base.h ist auch die Definition des Standard-Arrays FEDM_BYTE_ARRAY enthalten, der für alle Datencontainer verwendet wird.

6.1.1. Methoden (public)

Methode	Beschreibung
GetLibVersion	Gibt die Versionsnummer der Bibliothek als C-Zeichenkette zurück.
GetData	Die ausführende (überladene) Methode zum Lesen eines Parameterwertes aus einem Datencontainer. Diese Methode kann nur von der abgeleiteten Klasse FEDM_DataBase oder der wiederum davon abgeleiteten Leserklasse benutzt werden.
SetData	Die ausführende (überladene) Methode zum Schreiben eines Parameterwertes in einen Datencontainer. Diese Methode kann nur von der abgeleiteten Klasse FEDM_DataBase oder der wiederum davon abgeleiteten Leserklasse benutzt werden.
GetLanguage	Gibt die aktuell eingestellte Konstante für die Sprache zurück.
SetLanguage	Ändert die Sprache für Fehlertexte.
GetFeComFunction	Der Funktionszeiger einer Funktion der Bibliothek FECOM wird zurückgegeben. Wenn die Bibliothek noch nicht in den Adressraum eingebunden wurde, wird sie geladen und verbleibt anschließend im Adressraum. Im Destruktor wird die DLL wieder entladen. **WICHTIG: die Präprozessor-Definition _FEDM_COM_SUPPORT muss verwendet werden!**
GetFeUsbFunction	Der Funktionszeiger einer Funktion der Bibliothek FEUSB wird zurückgegeben. Wenn die Bibliothek noch nicht in den Adressraum eingebunden wurde, wird sie geladen und verbleibt anschließend im Adressraum. Im Destruktor wird die DLL wieder entladen. **WICHTIG: die Präprozessor-Definition _FEDM_USB_SUPPORT muss verwendet werden!**
GetFeTcpFunction	Der Funktionszeiger einer Funktion der Bibliothek FETCP wird zurückgegeben. Wenn die Bibliothek noch nicht in den Adressraum eingebunden wurde, wird sie geladen und verbleibt anschließend im Adressraum. Im Destruktor wird die DLL wieder entladen. **WICHTIG: die Präprozessor-Definition _FEDM_TCP_SUPPORT muss verwendet werden!**

6.2. FEDM_DataBase

Die Klasse FEDM_DataBase ist eine von FEDM_Base abgeleitete Basisklasse, die Datencontainer und auf die Datentypen der Datencontainer abgestimmte, überladene Zugriffsmethoden bereitstellt. Die Größe der Datencontainer ist 0. Die Größe wird erst in der von FEDM_DataBase abgeleiteten Leserklasse festgelegt.

Des Weiteren enthält die Klasse eine Reihe von abstrakten Methoden, die in jeder Leserklasse implementiert sein müssen. Abstrakte Methoden erlauben das Lesertyp-unabhängige Schreiben von Algorithmen, wenn man die Methoden von FEDM_DataBase anstelle der gleichlautenden Methoden der Leserklasse aufruft. Dadurch wird also eine typunabhängige Schnittstelle realisiert.

6.2.1. Attribute (public)

Attribut	Beschreibung	Leserfamilie	Organisation
FEDM_BYTE_ARRAY m_RFC_EEData	für Konfigurationsparameter des Lesers	ISC – RW – RO – RWA – TRI	16 Byte je Block
FEDM_BYTE_ARRAY m_RFC_RAMData	für temp. Konfigurations-parameter des Lesers	ISC	16 Byte je Block
FEDM_BYTE_ARRAY m_ACC_EEData	für Konfigurationsparameter des Lesers	ISC	32 Byte je Block
FEDM_BYTE_ARRAY m_ACC_RAMData	für temp. Konfigurations-parameter des Lesers	ISC	32 Byte je Block
FEDM_BYTE_ARRAY m_TmpData	für allgemeine temp. Protokolldaten	ISC - RW – RO – RWA - TRI	32 Byte je Block
FEDM_BYTE_ARRAY m_MjpData	für temp. Protokolldaten eines multijob-Polls	RWA – TRI	16 Byte je Block
FEDM_BYTE_ARRAY m_SN_Mem	für Seriennummer des Transponders	RW – RWA - TRI	16 Byte je Block
FEDM_BYTE_ARRAY m_ID_Mem	für ID-Nummern des Transponders	RW – RWA	16 Byte je Block
FEDM_BYTE_ARRAY m_AC_Mem	für Accountdaten des Transponders	RWA	16 Byte je Block
FEDM_BYTE_ARRAY m_PubMem	für public Datenblöcke des Transponders	RW – RWA – TRI	4 Byte je Block 16 Byte je Block für RWA
FEDM_BYTE_ARRAY m_SecMem	für secret Datenblöcke des Transponders	RW – RWA	4 Byte je Block
FEDM_BYTE_ARRAY m_ConfMem	für Konfigurationsdaten des Transponders	RW – RWA – TRI	4 Byte je Block
FEDM_BYTE_ARRAY m_DateMem	für Datumsdaten des Transponders	RWA - TRI	16 Byte je Block

6.2.2. Methoden (public)

Methode	Beschreibung
SetReaderHnd	Setzt den von der OBID-Funktionsbibliothek (FEISC, FERO, FERW, FERWA, FETRI, ...) erhaltenen Handle.
GetReaderHnd	Gibt den von der OBID-Funktionsbibliothek (FEISC, FERO, FERW, FERWA, FETRI, ...) erhaltenen Handle zurück.
SetBusAddress	Setzt die Busadresse für den Protokollverkehr.
GetBusAddress	Ermittelt die Busadresse des Protokollverkehrs.
GetFamilyCode	Ermittelt den Kurzstring für die Klassifizierung der Leserklasse.
GetReaderName	Gibt den Kurzstring zum Leser zurück.
GetReaderType	Gibt die ID zum Lesertyp zurück.
Serialize	Sub-Methode zur Realisierung der XML-Schnittstelle.
SetData	Die zentrale (überladene) Methode zum Schreiben eines Parameterwertes in einen Datencontainer. Der Aufruf wird an die Basisklasse FEDM_Base weitergeleitet, nachdem aus der Zugriffskonstanten der Typ des Datencontainers (Speichertyp-Konstante) ermittelt wurde. SetData unterstützt die Datentypen: bool, BOOL, UCHAR, UCHAR-Array, UINT, __int64, CString bzw AnsiString, STL-string und C-Zeichenkette. Eine zweite Variante erlaubt den Lesezugriff unter Angabe des exakten Indexes und der Speichertyp-Konstanten. Diese Variante unterstützt die Datenypen: UCHAR, UCHAR-Array, UINT, __int64 und CString bzw. AnsiString.
GetData	Die zentrale (überladene) Methode zum Lesen eines Parameterwertes aus einem Datencontainer. Der Aufruf wird an die Basisklasse FEDM_Base weitergeleitet, nachdem aus der Zugriffskonstanten der Typ des Datencontainers (Speichertyp-Konstante) ermittelt wurde. SetData unterstützt die Datentypen: bool, BOOL, UCHAR, UCHAR-Array, UINT, __int64, CString bzw AnsiString, STL-string und C-Zeichenkette. Eine zweite Variante erlaubt den Schreibzugriff unter Angabe des exakten Indexes und der Speichertyp-Konstanten. Diese Variante unterstützt die Datenypen: UCHAR, UCHAR-Array, UINT, __int64 und CString bzw. AnsiString.

6.2.3. Abstrakte Methoden (public)

Methode	Beschreibung
EvalLibDependencies	Methode verifiziert die Kompatibilität mit abhängigen Funktionsbibliotheken
SendProtocol	Die zentrale Kommunikationsmethode.
FindBaudRate	Ermittelt Baudrate und Protokoll-Rahmen des Lesers und stellt Schnittstelle ein.
GetLastProt	Methode zum Abrufen des letzten Sende- oder Empfangs-Protokolls.
SetPortHnd	Transferiert den von der in der Protokolltransfer-Schicht liegenden Funktionsbibliothek (FECOM, FEUSB, ...) erhaltenen Handle in die OBID-Funktionsbibliothek (FEISC, FERW, FERWA, FETRI, ...).
GetPortHnd	Ermittelt den von der in der Protokolltransfer-Schicht liegenden Funktionsbibliothek (FECOM, FEUSB, ...) erhaltenen Handle aus der OBID-Funktionsbibliothek (FEISC, FERW, FERWA, FETRI, ...).
SetProtocolFrameSupport	Stellt den Protokolltyp für den Protokollverkehr ein.
GetProtocolFrameSupport	Ermittelt den Protokolltyp für den Protokollverkehr.

Methode	Beschreibung
SetReaderType	Methode bekommt als Parameter die ID des Lesertyps übergeben und setzt interne Variablen für diesen Lesertyp.
GetLastError	Gibt letzten Fehlercode zurück.
GetLastStatus	Gibt Statuswert des letzten Protokolls zurück.
GetErrorText	Gibt zu jedem Fehlercode den zugehörigen Fehlertext zurück.
GetStatusText	Gibt zu jedem Statusbyte den zugehörigen Statustext zurück.
Serialize	Hauptmethode für Serialisierung. Ermöglicht das Serialisieren der Containerdaten in Dateien. Zwei Versionen sind implementiert: eine Version für den Dateityp XML und eine zweite Version für MFC-basierte Applikationen. Anm: Letztere Variante ist in der Leserklasse für die OBID i-*scan*® Familie nicht mehr implementiert.
SerializeIn	Submethode zur Realisierung der XML-Schnittstelle.
SerializeOut	Submethode zur Realisierung der XML-Schnittstelle.
GetTableData	Die zentrale (überladene) Methode zum Lesen eines Parameterwertes oder von Datenblöcken aus einer Tabelle. Diese Variante unterstützt die Datenypen: bool, UCHAR, UCHAR-Array, UINT, __int64, CString bzw. AnsiString und STL-string.
SetTableData	Die zentrale (überladene) Methode zum Schreiben eines Parameterwertes oder von Datenblöcken in eine Tabelle. Diese Variante unterstützt die Datenypen: bool, UCHAR, UCHAR-Array, UINT, __int64, CString bzw. AnsiString und STL-string.
FindTableIndex	Die zentrale (überladene) Methode zum Ermitteln des Tabellen-Indexes anhand eines Wertes. Diese Variante unterstützt die Datenypen: bool, UCHAR, UINT, __int64 CString bzw. AnsiString und STL-string.
VerifyTableData	Vergleich von gesendeten mit empfangenen Transponderdaten (z.Zt. nur OBID i-*scan*® und OBID® *classic-pro* Leserfamilie)
GetTableSize	Ermittelt die Größe einer Tabelle.
SetTableSize	Stellt die Größe einer Tabelle ein.
GetTableLength	Ermittelt die Anzahl gültiger Tabelleneinträge.
ResetTable	Initialisiert eine Tabelle komplett oder einzelne Elemente darin.

6.3. FEDM_XMLBase

Die Klasse FEDM_XMLBase ist die Basisklasse für die Serialisierung von Objektdaten im XML-Format. In ihr sind alle Basismethoden, eine Reihe von Attributen und der Objektbaum für die XML-Struktur enthalten.

FEDM_XMLBase ist keine allgemeine XML-Klasse für beliebige Dateien. Sondern diese Klasse ist speziell zugeschnitten für die spezialisierte Aufgabe der Serialisierung von Leserkonfigurationen oder ähnlichen Objektdaten. Daher ist sie auch klein im Gegensatz zu den mächtigen kommerziellen oder nicht-kommerziellen XML-Bibliotheken.

WICHTIG: Die XML-Serialisierungsklassen werden nur kompiliert, wenn die Präprozessor-Definition **_FEDM_XML_SUPPORT** gesetzt ist!

6.3.1. Methoden (public)

Methode	Beschreibung
OpenDoc	Öffnet ein XML-Dokument.
CloseDoc	Schließt ein XML-Dokument.
ReadDoc	Liest den kompletten Inhalt des XML-Dokuments.
WriteDoc	Schreibt die Unicode-Strings in das Dokument.
IsXmlDoc	Prüft, ob die mit OpenDoc geöffnete Datei ein XML-Dokument ist.
HasOBIDTag	Prüft, ob die mit OpenDoc geöffnete Datei ein OBID-Tag enthält.
BuildTag	Erzeugt eine neue Tag-Struktur
AddTagValue	Fügt den Inhalt zu einem Tag hinzu.
AddTagAttrib	Fügt ein Attribut mit Wert einem Tag hinzu.
AddTagItem	Fügt die mit BuildTag erzeugte Tag-Struktur in den internen XML-Baum ein.
FindTag	Sucht einen Tag im internen XML-Baum.
GetTagValue	Gibt den Inhalt eines Tags zurück.
GetTagAttrib	Gibt ein Attribut mit Wert eines Tags zurück.
GetLastError	Gibt den Inhalt der internen Fehlervariable m_iLastError zurück.

6.4. FEDM_XMLReaderCfgDataModul

Die Klasse FEDM_XMLReaderCfgDataModul ist von der Basisklasse FEDM_XMLBase abgeleitet und speziell für die Serialisierung der Leserkonfigurationsdaten aus den in der Leserklasse FEDM_DataBase implementierten Datencontainer m_RFC_EEData und m_RFC_RAMData, bzw. m_ACC_EEData und m_ACC_RAMData entwickelt worden.

6.4.1. Methoden (public)

Methode	Beschreibung
SerializeIn	Sub-Methode liest gesamtes XML-Dokument ein und speichert die Leserkonfigurationsdaten in den Datencontainern der Leserklasse FEDM_DataBase.
SerializeOut	Sub-Methode schreibt gesamtes XML-Dokument mit Header und Leserkonfigurationsdaten aus den Datencontainern der Leserklasse FEDM_DataBase.
QueryDocType	Öffnet ein XML-Dokument und ermittelt den Dokumenten-Typ (z.Zt. nur "Reader Configuration File")
QueryDocVersion	Öffnet ein XML-Dokument und ermittelt die Dokumenten-Version (z.B. "1.0")
QueryReaderFamily	Öffnet ein XML-Dokument und ermittelt die Leser-Familie aus dem Tag "reader-family".
QueryReaderType	Öffnet ein XML-Dokument und ermittelt den Lesertyp aus dem Tag "reader-type".
GetComment	Gibt den Kommentartext aus der Variablen m_wsComment zurück.
SetComment	Speichert den Kommentartext in der Variable m_wsComment. Ist ein Kommentartext vorhanden, wird dieser in SerializeOut in das XML-Dokument geschrieben.
GetPrgName	Gibt den Programmnamen aus der Variablen m_wsPrgName zurück.
SetPrgName	Speichert den Programmnamen in der Variablen m_wsPrgName. Ist ein Programmname vorhanden, wird dieser in SerializeOut in das XML-Dokument geschrieben.
GetPrgVer	Gibt die Programmversion aus der Variablen m_wsPrgVer zurück.
SetPrgVer	Speichert die Programmversion in der Variablen m_wsPrgVer. Ist eine Programmversion vorhanden, wird dieser in SerializeOut in das XML-Dokument geschrieben.
GetHost	Gibt die TCP/IP-Hostadresse aus der Variablen m_wsHost zurück. Die TCP/IP Hostadresse wird in die XML-Datei des Programms ISOStart eingefügt.
GetPort	Gibt die TCP/IP Portnummer aus der Variablen m_wsPort zurück. Die TCP/IP Portnummer wird in die XML-Datei des Programms ISOStart eingefügt.
GetCommMode	Gibt den Kommunikationsmodus aus der Variablen m_wsCommMode zurück. Der Kommunikationsmodus ("Serial", "USB", "TCP") wird in die XML-Datei des Programms ISOStart eingefügt.

7. Globale Funktionen

7.1. FEDM_Functions

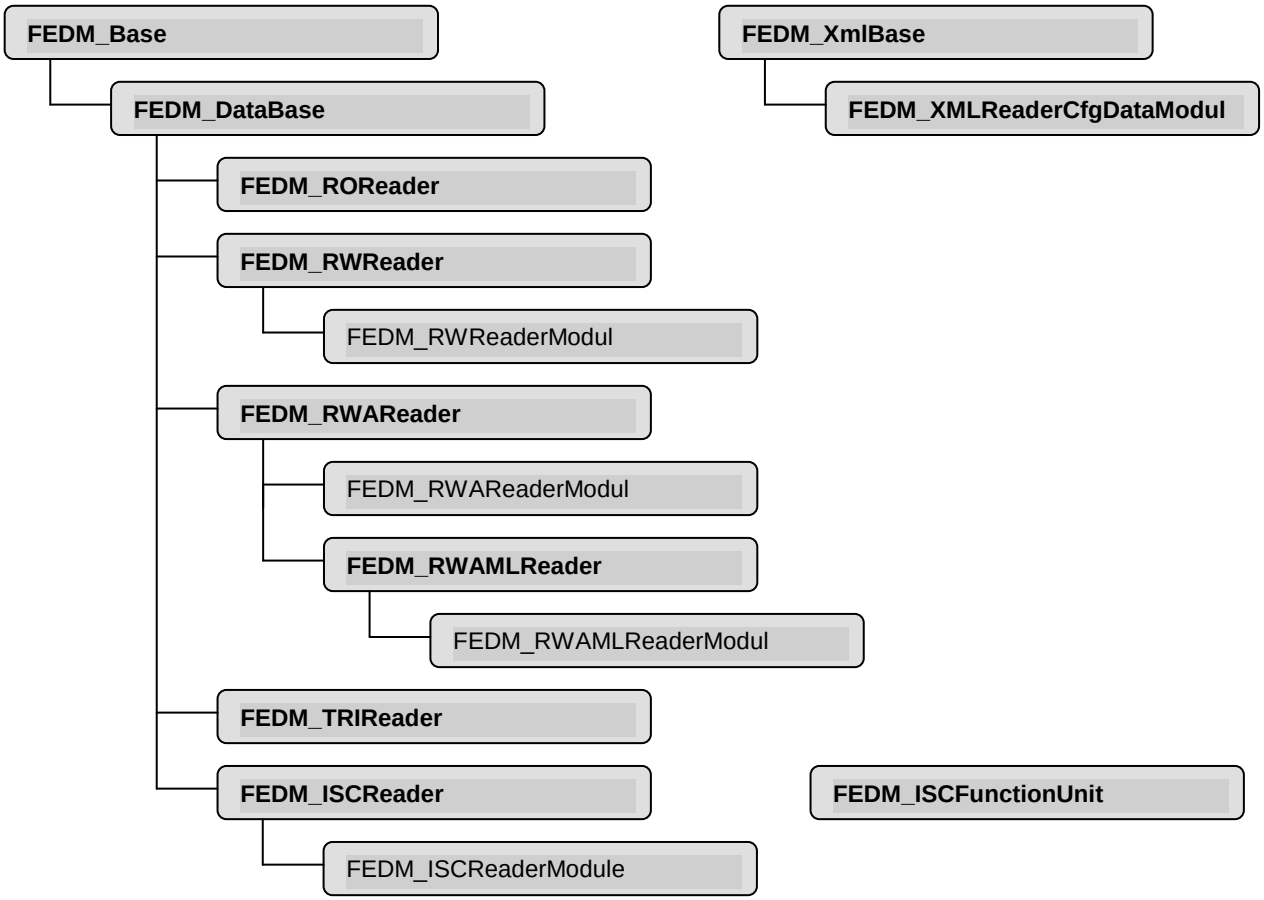
Funktion	Beschreibung
FEDM_GetMemIDOfID	Ermittelt die Speichertyp-Konstante aus der Zugriffskonstanten (s. Kapitel 5.3. Zugriffskonstanten für temporäre Protokolldaten). Beispiel: FEDM_GetMemIDOfID(FEDM_ISCM_EE_COM_BUSADR) gibt den Wert 3 zurück.
FEDM_GetAdrOfID	Ermittelt die Adresse eines Parameters aus der Zugriffskonstanten.
FEDM_GetByteCntOfID	Ermittelt die Anzahl der Bytes, aus denen ein Parameter besteht, aus der Zugriffskonstanten.
FEDM_ToRAM	Zugriff auf Parameter im Container m_RFC_RAMData mit der Zugriffskonstanten für den Container m_RFC_EEData.
FEDM_MdfyMemID	Modifiziert die Speichertyp-Konstante der Zugriffskonstanten.
FEDM_MdfyBlockNr	Modifiziert die Blocknummer der Zugriffskonstanten.
FEDM_AddOff2BlockNr	Addiert Offset zur Blocknummer der Zugriffskonstanten hinzu.
FEDM_ConvHexCharToInt	Konvertiert eine C-Zeichenkette mit einem hexadezimalen String in einen Integer-Wert. Der String darf maximal 8 Zeichen enthalten. Alle Zeichen außer 0..9, a..f, A..F werden vorher entfernt. Beispiel: sIn = "1122F05E" -> iOut = 287502430
FEDM_ConvHexCharToUInt	Konvertiert eine C-Zeichenkette mit einem hexadezimalen String in einen vorzeichenlosen Integer-Wert. Der String darf maximal 8 Zeichen enthalten. Alle Zeichen außer 0..9, a..f, A..F werden vorher entfernt. Beispiel: sIn = "1122F05E" -> uiOut = 287502430
FEDM_ConvHexCharToInt64	Konvertiert eine C-Zeichenkette mit einem hexadezimalen String in einen 64-Bit Integer-Wert. Der String darf maximal 16 Zeichen enthalten. Alle Zeichen außer 0..9, a..f, A..F werden vorher entfernt. Beispiel: sIn = "1122F05E1122F05E" -> i64Out = 1234813534658031710
FEDM_ConvHexCharToUChar	Konvertiert eine C-Zeichenkette mit einem hexadezimalen String in eine C-Zeichenkette. Alle Zeichen außer 0..9, a..f, A..F werden vorher entfernt. Beispiel: sIn = “1122F05E" --> ucOutBuf = {0x11, 0x22, 0xF0, 0x5E}
FEDM_ConvHexStrToInt	Konvertiert einen hexadezimalen String in einen Integer-Wert. Der String darf maximal 8 Zeichen enthalten. Alle Zeichen außer 0..9, a..f, A..F werden vorher entfernt. Beispiel: sIn = "1122F05E" -> iOut = 287502430
FEDM_ConvHexStrToUInt	Konvertiert einen hexadezimalen String in einen vorzeichenlosen Integer-Wert. Der String darf maximal 8 Zeichen enthalten. Alle Zeichen außer 0..9, a..f, A..F werden vorher entfernt. Beispiel: sIn = "1122F05E" -> uiOut = 287502430
FEDM_ConvHexStrToInt64	Konvertiert einen hexadezimalen String in einen 64-Bit Integer-Wert. Der String darf maximal 16 Zeichen enthalten. Alle Zeichen außer 0..9, a..f, A..F werden vorher entfernt. Beispiel: sIn = "1122F05E1122F05E" -> i64Out = 1234813534658031710

Funktion	Beschreibung
FEDM_ConvHexStrToUChar	Konvertiert einen hexadezimalen String in eine C-Zeichenkette. Alle Zeichen außer 0..9, a..f, A..F werden vorher entfernt. Beispiel: sIn = “1122F05E" --> ucOutBuf = {0x11, 0x22, 0xF0, 0x5E}
FEDM_ConvHexUCharToInt	Konvertiert eine C-Zeichenkette in einen Integer-Wert. Die C-Zeichenkette darf maximal 4 Zeichen enthalten. Beispiel: ucInBuf = {0x11, 0x22, 0xF0, 0x5E} -> iOut = 287502430
FEDM_ConvHexUCharToUInt	Konvertiert eine C-Zeichenkette in eine n vorzeichenlosen Integer-Wert. Die C-Zeichenkette darf maximal 4 Zeichen enthalten. Beispiel: ucInBuf = {0x11, 0x22, 0xF0, 0x5E} -> uiOut = 287502430
FEDM_ConvHexUCharToInt64	Konvertiert eine C-Zeichenkette in einen 64-Bit Integer-Wert. Die C-Zeichenkette darf maximal 8 Zeichen enthalten. Beispiel: ucInBuf = {0x11, 0x22, 0xF0, 0x5E, 0x11, 0x22, 0xF0, 0x5E } -> iOut = 1234813534658031710
FEDM_ConvHexUCharToHexChar	Konvertiert eine C-Zeichenkette in eine C-Zeichenkette mit hexadezimalen String. Beispiel: ucInBuf = {0x11, 0x22, 0xF0, 0x5E} -> sOut = “1122F05E“
FEDM_ConvHexUCharToHexStr	Konvertiert eine C-Zeichenkette in einen hexadezimalen String. Beispiel: ucInBuf = {0x11, 0x22, 0xF0, 0x5E} -> sOut = “1122F05E“
FEDM_ConvHexUCharToBcdChar	Konvertiert ein Byte-Array in ein BCD-Array. Beispiel: ucInBuf = {0x02, 0x01, 0x00, 0x00, 0x01, 0x04} -> ucOutBuf = {0x21, 0x00, 0x14}
FEDM_ConvBcdCharToHexUChar	Konvertiert ein BCD-Array in ein Byte-Array. Beispiel: ucInBuf = {0x21, 0x00, 0x14} -> ucOutBuf = {0x02, 0x01, 0x00, 0x00, 0x01, 0x04}
FEDM_ConvIntToHexStr	Konvertiert einen Integer-Wert in einen hexadezimalen String. Beispiel: iIn = 287502430 -> sOut = "1122F05E"
FEDM_ConvIntToHexUChar	Konvertiert einen Integer-Wert in eine C-Zeichenkette. Beispiel: iIn = 287502430 -> ucOutBuf = {0x11, 0x22, 0xF0, 0x5E}
FEDM_ConvIntToHexChar	Konvertiert einen Integer-Wert in eine C-Zeichenkette mit hexadezimalen String. Beispiel: iIn = 287502430 -> sOut = “1122F05E“
FEDM_ConvUIntToHexStr	Konvertiert einen vorzeichenlosen Integer-Wert in einen hexadezimalen String. Beispiel: uiIn = 287502430 -> sOut = "1122F05E"
FEDM_ConvUIntToHexUChar	Konvertiert einen vorzeichenlosen Integer-Wert in eine C-Zeichenkette. Beispiel: uiIn = 287502430 -> ucOutBuf = {0x11, 0x22, 0xF0, 0x5E}
FEDM_ConvUIntToHexChar	Konvertiert einen vorzeichenlosen Integer-Wert in eine C-Zeichenkette mit hexadezimalen String. Beispiel: uiIn = 287502430 -> sOut = “1122F05E“
FEDM_ConvInt64ToHexStr	Konvertiert einen 64-Bit Integer-Wert in einen hexadezimalen String. Beispiel: i64In = 1234813534658031710 -> sOut = "1122F05E1122F05E"
FEDM_ConvInt64ToHexUChar	Konvertiert einen 64-Bit Integer-Wert in eine C-Zeichenkette. Beispiel: i64In = 1234813534658031710 -> iOutBuf = {0x11, 0x22, 0xF0, 0x5E, 0x11, 0x22, 0xF0, 0x5E }

Funktion	Beschreibung
FEDM_ConvInt64ToHexChar	Konvertiert einen 64-Bit Integer-Wert in eine C-Zeichenkette mit hexadezimalen String. Beispiel: i64In = 1234813534658031710 -> sOut = " 1122F05E1122F05E"
FEDM_ConvTwoAsciiToUChar	Baut aus zwei ASCII-Zeichen (0..9, a..f, A..F) ein char-Wert zusammen.
FEDM_RemNoHexChar	Entfernt alle nicht-ASCII-Zeichen aus einem String und kopiert Ergebnis in eine C-Zeichenkette.
FEDM_IsHex	Testet String auf ASCI-Zeichen (0..9, a..f, A..F).

8. Anhang

8.1. Klassenbaum



8.2. Liste der Fehlercodes

Alle nachfolgend aufgelisteten Konstanten sind in der Datei FEDM.h deklariert.

Fehler-Konstante	Wert	Beschreibung
FEDM_MODIFIED	1	Kein Fehler aufgetreten, Container wurde modifiziert
FEDM_OK	0	Kein Fehler aufgetreten
FEDM_ERROR_BLOCK_SIZE	-101	Die Blockgröße in der Zugriffskonstanten ist falsch
FEDM_ERROR_BIT_BOUNDARY	-102	Die Bitgrenze in der Zugriffskonstanten ist falsch
FEDM_ERROR_BYTE_BOUNDARY	-103	Die Bytegrenze in der Zugriffskonstanten ist falsch
FEDM_ERROR_ARRAY_BOUNDARY	-104	Die Arraygrenze eines Datencontainers wurde überschritten
FEDM_ERROR_BUFFER_LENGTH	-105	Die Länge des Datenpuffers ist nicht ausreichend
FEDM_ERROR_PARAMETER	-106	Ein Übergabeparameter ist unbekannt
FEDM_ERROR_STRING_LENGTH	-107	Der übergebene String ist zu lang
FEDM_ERROR_ODD_STRING_LENGTH	-108	Der übergebene String enthält eine ungerade Anzahl Zeichen
FEDM_ERROR_NO_DATA	-109	Keine Daten im Protokoll
FEDM_ERROR_NO_READER_HANDLE	-110	Kein Reader-Handle gesetzt
FEDM_ERROR_NO_PORT_HANDLE	-111	Kein Port-Handle gesetzt
FEDM_ERROR_UNKNOWN_CONTROL_BYTE	-112	Unbekanntes Steuerbyte
FEDM_ERROR_UNKNOWN_MEM_ID	-113	Unbekannte Speicher-ID
FEDM_ERROR_UNKNOWN_POLL_MODE	-114	Unbekannter Pollmode
FEDM_ERROR_NO_TABLE_DATA	-115	Keine Daten in einer Tabelle
FEDM_ERROR_UNKNOWN_ERROR_CODE	-116	Unbekannter Fehlercode
FEDM_ERROR_UNKNOWN_COMMAND	-117	Unbekanntes Kommando
FEDM_ERROR_UNSUPPORTED	-118	Funktion wird nicht unterstützt
FEDM_ERROR_NO_MORE_MEM	-119	Kein Programmspeicher meht verfügbar
FEDM_ERROR_NO_READER_FOUND	-120	Kein Leser gefunden
FEDM_ERROR_NULL_POINTER	-121	Der übergebene Zeiger ist NULL
FEDM_ERROR_UNKNOWN_READER_TYPE	-122	Unbekannter Lesertyp
FEDM_ERROR_UNSUPPORTED_READER_TYPE	-123	Funktion kann für diesen Lesertyp nicht verwendet werden
FEDM_ERROR_UNKNOWN_TABLE_ID	-124	Unbekannte Tabellenkonstante
FEDM_ERROR_UNKNOWN_LANGUAGE	-125	Unbekannte Sprachenkonstante

Fehler-Konstante	Wert	Beschreibung
FEDM_ERROR_NO_TABLE_SIZE	-126	Tabelle hat Größe 0
FEDM_ERROR_SENDBUFFER_OVERFLOW	-127	Sendepuffer ist voll
FEDM_ERROR_VERIFY	-128	Daten sind ungleich
FEDM_ERROR_OPEN_FILE	-129	Fehler beim Öffnen der Datei
FEDM_ERROR_SAVE_FILE	-130	Fehler beim Speichern der Datei
FEDM_ERROR_UNKNOWN_TRANSPONDER_TYPE	-131	unbekannter Transpondertyp
FEDM_ERROR_READ_FILE	-132	Fehler beim Lesen der Datei
FEDM_ERROR_WRITE_FILE	-133	Fehler beim Schreiben in Datei
FEDM_ERROR_UNKNOWN_EPC_TYPE	-134	unbekannter EPC-Typ
FEDM_ERROR_UNSUPPORTED_PORT_DRIVER	-135	Funktion unterstützt nicht den aktiven Kommunikations-Treiber
FEDM_ERROR_UNKNOWN_ADDRESS_MODE	-136	unbekannter Adressmodus
FEDM_ERROR_ALREADY_CONNECTED	-137	Leserobjekt ist bereits mit einem Port verbunden
FEDM_ERROR_NOT_CONNECTED	-138	Leserobjekt ist nicht mit einem Port verbunden
FEDM_ERROR_NO_MODULE_HANDLE	-139	Kein Handle eines Lesermoduls gefunden
FEDM_ERROR_EMPTY_MODULE_LIST	-140	Die Modulliste ist leer
FEDM_ERROR_MODULE_NOT_FOUND	-141	Modul nicht in Modulliste gefunden
FEDM_ERROR_DIFFERENT_OBJECTS	-142	Laufzeitobjekte sind nicht identisch
FEDM_ERROR_NOT_AN_EPC	-143	Seriennummer des Transponders ist keine EPC
FEDM_ERROR_OLD_LIB_VERSION	-144	Veraltete Bibliothek erkannt (Fehlercode für Java/.NET-Bibliotheken)
FEDM_ERROR_WRONG_READER_TYPE	-145	Falscher Lesertyp
FEDM_ERROR_CRC	-146	CRC-Fehler in Datei
FEDM_ERROR_CFG_BLOCK_PREVIOUSLY_NOT_READ	-147	Konfigurationsblock muss erst gelesen werden
FEDM_ERROR_UNSUPPORTED_CONTROLLER_TYPE	-148	Nicht unterstützter Controller-Typ
FEDM_ERROR_VERSION_CONFLICT	-149	Versionskonflikt mit einer oder mehrerer abhängiger Bibliotheken
FEDM_ERROR_UNSUPPORTED_NAMESPACE	-150	Namespace wird vom Lesertyp nicht unterstützt.
FEDM_ERROR_TASK_STILL_RUNNING	-151	Asynchroner Task ist noch am laufen
FEDM_ERROR_TAG_HANDLER_NOT_IDENTIFIED	-152	TagHandler-Typ konnte nicht identifiziert werden
FEDM_ERROR_UNVALID_IDD_LENGTH	-153	Längenangabe für IDD ist außerhalb des zulässigen Bereichs
FEDM_ERROR_UNVALID_IDD_FORMAT	-154	Formateinstellung für IDD ist falsch
FEDM_ERROR_UNKNOWN_TAG_HANDLER_TYPE	-155	Unbekannter TagHandler-Typ

Fehler-Konstante	Wert	Beschreibung
FEDM_ERROR_UNSUPPORTED_TRANSPONDER_TYPE	-156	Transponder- oder Chiptyp wird nicht unterstützt
FEDM_ERROR_CONNECTED_WITH_OTHER_MODULE	-157	Nur TCP/IP: eine Verbindung zum gleichen Leser wurde bereits von einem anderen Modul aufgebaut.
FEDM_ERROR_INVENTORY_NO_TID_IN_UID	-158	Inventory mit Rückgabe von UID = EPC + TID, aber TID fehlt
FEDM_XML_ERROR_NO_XML_FILE	-200	Datei ist kein XML-Dokument
FEDM_XML_ERROR_NO_OBID_TAG	-201	Datei enthält kein Element 'OBID'
FEDM_XML_ERROR_NO_CHILD_TAG	-202	Kein Sub-Element vorhanden
FEDM_XML_ERROR_TAG_NOT_FOUND	-203	Element nicht in Dokument
FEDM_XML_ERROR_DOC_NOT_WELL_FORMED	-204	XML-Dokument nicht wohlgeformt
FEDM_XML_ERROR_NO_TAG_VALUE	-205	Kein Element-Inhalt vorhanden
FEDM_XML_ERROR_NO_TAG_ATTRIBUTE	-206	Kein Attribute vorhanden
FEDM_XML_ERROR_DOC_FILE_VERSION	-207	Ungültige Dokument-Version
FEDM_XML_ERROR_DOC_FILE_FAMILY	-208	Dokument für andere Leserfamilie
FEDM_XML_ERROR_DOC_FILE_TYPE	-209	Falscher Dateityp
FEDM_XML_ERROR_WRONG_CONTROLLER_TYPE	-210	Falscher Controller-Typ
FEDM_XML_ERROR_WRONG_MEM_BANK_TYPE	-211	Falsche Speicherbank

8.3. Liste der Leserfamilien

Alle nachfolgend aufgelisteten Konstanten sind in der Datei FEDM.h deklariert.

Leser-Konstante	Wert	Beschreibung
FEDM_RW_FAMILY	1	
FEDM_RWA_FAMILY	2	
FEDM_ISC_FAMILY	3	
FEDM_TRI_FAMILY	4	
FEDM_RO_FAMILY	5	

8.4. Liste der Sprachen-Konstanten

Alle nachfolgend aufgelisteten Konstanten sind in der Datei FEDM.h deklariert.

Sprachen-Konstante	Wert	Beschreibung
FEDM_LANG_GERMAN	7	
FEDM_LANG_ENGLISH	9	Voreinstellung

8.5. Liste der Protokolltyp-Konstanten

Alle nachfolgend aufgelisteten Konstanten sind in der Datei FEDM.h deklariert.

Protokolltyp-Konstante	Wert	Beschreibung
FEDM_PRT_FRAME_STANDARD	1	Voreinstellung
FEDM_PRT_FRAME_ADVANCED	2	

8.6. Liste der Speichertyp-Konstanten

Alle nachfolgend aufgelisteten Konstanten sind in der Datei FEDM.h deklariert.

Speichertyp-Konstante	Wert	Beschreibung
FEDM_RFC_EEDATA_MEM	3	Für Konfigurationsparameter des Lesers
FEDM_RFC_RAMDATA_MEM	4	Für temp. Konfigurationsparameter des Lesers
FEDM_ACC_EEDATA_MEM	5	Für Konfigurationsparameter des Lesers

Speichertyp-Konstante	Wert	Beschreibung
FEDM_ACC_RAMDATA_MEM	6	Für temp. Konfigurationsparameter des Lesers
FEDM_TMPDATA_MEM	10	Für allgemeine temp. Protokolldaten
FEDM_MJPDATA_MEM	11	Für temp. Protokolldaten eines multijob-Polls
FEDM_SN_MEM	20	Für Seriennummer des Transponders
FEDM_ID_MEM	21	Für ID-Nummern des Transponders
FEDM_AC_MEM	22	Für Accountdaten des Transponders
FEDM_PUB_MEM	23	Für public Datenblöcke des Transponders
FEDM_SEC_MEM	24	Für secret Datenblöcke des Transponders
FEDM_CONF_MEM	25	Für Konfigurationsdaten des Transponders
FEDM_DATE_MEM	26	Für Datumsdaten des Transponders

8.7. Makros

Alle nachfolgend aufgelisteten Makros sind in der Datei FEDM.h definiert.

Makro	Beschreibung
FEDM_CHK1	Überprüft, ob der Rückgabewert einer Funktion negativ ist. Makro kann nur in Readerklasse verwendet werden.
FEDM_CHK2	Überprüft, ob der Rückgabewert einer Funktion negativ ist und setzt in diesem Fall die globale Fehlervariable in FEDM_TMPDATA_MEM. Makro kann nur in Readerklasse verwendet werden.
FEDM_CHK3	Überprüft einen Pointer auf NULL.
FEDM_CHK4	Überprüft, ob der Rückgabewert einer Funktion ungleich 0 ist und setzt in diesem Fall die globale Fehlervariable in FEDM_TMPDATA_MEM. Makro kann nur in Readerklasse verwendet werden.
FEDM_CHK5	Überprüft einen Pointer auf NULL und setzt in diesem Fall die globale Fehlervariable in FEDM_TMPDATA_MEM. Makro kann nur in Readerklasse verwendet werden.
FEDM_CHK6	Überprüft, ob der Rückgabewert einer Funktion negativ ist. Makro kann nur in Readerklasse verwendet werden.
FEDM_CHK7	Überprüft, ob der Rückgabewert einer Funktion negativ ist und setzt in diesem Fall die globale Fehlervariable in FEDM_TMPDATA_MEM und gibt NULL zurück. Makro kann nur in Readerklasse verwendet werden.
FEDM_CHK8	Überprüft, ob der Wert einer Funktion null oder negativ ist und setzt in diesem Fall die globale Fehlervariable auf FEDM_ERROR_STRING_LENGTH. Makro kann nur in Readerklasse verwendet werden.
FEDM_RETURN	Setzt die globale Fehlervariable in FEDM_TMPDATA_MEM. Makro kann nur in Readerklasse verwendet werden.
FEDM_IS_COMPORT	Überprüft den Porthandle, ob es ein Handle eines seriellen COM-Ports ist.
FEDM_IS_USBPORT	Überprüft den Porthandle, ob es ein Device-Handle eines USB-Ports ist.
FEDM_IS_TCPPORT	Überprüft den Porthandle, ob es ein Socket-Handle eines TCP/IP-Ports ist.

8.8. Änderungshistorie

V4.06.01

- Linux:

  Version für 64-Bit

V4.05.00

- Neue globale Hilfsfunktionen:

  FEDM_ConvBcdCharToHexUChar und **FEDM_ConvHexUCharToBcdChar**

V4.03.00

- Neuer Fehlercode: FEDM_ERROR_INVENTORY_NO_TID_IN_UID

V4.02.00

- Diese neue Version ist nicht vollständig kompatibel zur Vorversion. Weitere Details finden sich in der Dokumentation zum Teil B

- Windows:

  1. Migration der Entwicklungsumgebung von Visual Studio 2005 zu Visual Studio 2010.
  2. Erstes Release der 64-Bit Version
  3. Anbindung an Log-Manager

- Erste Release-Version für Mac OS X, ab V10.7.3

V4.00.00

- Diese neue Version ist nicht vollständig kompatibel zur Vorversion. Weitere Details finden sich in der Dokumentation zum Teil B

V3.03.00

- Diese neue Version ist möglicherweise nicht vollständig kompatibel zur Vorversion. Dies hängt von der Verwendung von Klassen, Methoden und Konstanten ab.
- Weitere Details finden sich in der Dokumentation zum Teil B

V3.01.00

- Diese neue Version ist möglicherweise nicht vollständig kompatibel zur Vorversion. Dies hängt von der Verwendung von Klassen, Methoden und Konstanten ab.
- USB-Support für Windows CE
- Neue Fehlercodes

V3.00.00

- Diese neue Version ist möglicherweise nicht vollständig kompatibel zur Vorversion. Dies hängt von der Verwendung von Klassen, Methoden und Konstanten ab.
- USB-Support für Linux
- Bibliothek als dynamische Link-Bibliothek (DLL/SO) verfügbar
- Ablage des Quellcodes in neuer Verzeichnisstruktur
- Nur noch eine Include-Datei
- Neue Fehlercodes
- Die Zugriffskonstanten für Konfigurationsparameter sind ersetzt worden durch den Namespace ReaderConfig (z. Zt. nur für OBID i-*scan*® Familie)
- Umbenennung folgender Konstanten

Alte Konstante	Neue Konstante
FEDM_EEDATA_MEM	FEDM_RFC_EEDATA_MEM
FEDM_RAMDATA_MEM	FEDM_RFC_RAMDATA_MEM

- Umbenennung folgender Variablen in der Klasse FEDM_DataBase

Alte Bezeichnung	Neue Bezeichnung
EEData	m_RFC_EEData
RAMData	m_RFC_RAMData
TmpData	m_TmpData

V2.05.06

- Die Linux-Version wurde mit dem Compiler GCC 3.3.3 unter SuSE Linux 9.1 erstellt

V2.05.01

- Modifizierte Lizenzbestimmungen

V2.05.00

- Neue Leserklasse FEDM_ISCReaderModule (s. Teil B der Dokumentation)

- Neue globale Konvertierungsfunktionen für HexChar (FEDM_ConvHexCharTo...)

V2.04.00

- Optional statische Bindung mit Präprozessor-Definition **_FEDM_SUPPORT_SLINK** möglich

V2.03.00

- Änderung der Blockgröße des Byte-Arrays TmpData für temporäre Daten von 16 auf 32. Dies hat nur interne Auswirkungen.
- Neue Präprozessor-Definition **_FEDM_XML_SUPPORT** zur Einbindung der XML-Serialisierungsklassen. Diese Option war in früheren Versionen nicht notwendig. Vor der Neukompilation mit der aktuellen Version muss also die Definition gesetzt werden, wenn die XML-Serialisierungsklassen genutzt werden.
- Neue Präprozessor-Definition **_FEDM_MFC_SUPPORT** zur Einbindung der MFC-Klassen CString und CArchive. Diese Option war in früheren Versionen nicht notwendig. Vor der Neukompilation mit der aktuellen Version muss also die Definition gesetzt werden, wenn die genannten MFC-Klassen genutzt werden.

V2.02.00

- keine Änderungen in den Basisklassen.

V2.01.00

- keine Änderungen in den Basisklassen.

V2.00.00

- keine Änderungen in den Basisklassen.

V1.09.10

- Neue Fehlercodes.
- Unterstützung für Advanced Protocol Frame mit zwei Längenbytes.

V1.08.00

- Neue Klassen FEDM_XMLBase und FEDM_XMLReaderCfgDataModul realisieren Schnittstelle zur Serialisierung von Leser-Konfigurationen im XML-Format.
- Funktion FEDM_DataBase::GetFamilyCode wurde umgestellt und ist nicht mehr kompatibel zur Vorversion

- Neue Funktionen in abstrakter Basisklasse FEDM_DataBase: GetReaderName, GetReaderType, SetReaderType und drei neue Serialize-Funktionen.
- Unterstützung für USB-Leser.
- Unterstützung der TCP/IP-Schnittstelle.
- Umstellung auf dynamische Bindung der Kommunikations-Bibliotheken (FECOM, FETCP, FEUSB) mittels Funktionszeigen. Dadurch neue Funktionen in der Basisklasse FEDM_Base: GetFeComFunction, GetFeTcpFunction, GetFeUsbFunction
- Unterstützung des Datentyps bool, __int64 und STL-string von den Funktionen GetTableData, SetTableData, FindTableData.
- Flexible Einbindung der Kommunikations-Bibliotheken (FECOM, FETCP, FEUSB) per Präprozessor-Definitionen.
- Flexible Kompilation für Windows oder Linux per Präprozessor-Definitionen.
- Neue Fehlercodes.

V1.06.00

- Neue Klassen FEDM_RWAMLReader, FEDM_RWReaderModul, FEDM_RWAReaderModul und FEDM_RWAMLReaderModul.
- Unterstützung für GNU C-Compiler unter Linux.

V1.05.00

- interne Zwischenversion.

V1.04.00

- Neue Klassen für Read-Only Leserfamilie und für megalock Produktpalette.

V1.03.00

- Sprachenunterstützung mit den neuen Funktionen SetLanguage und GetLanguage. Die Sprachen Deutsch und Englisch sind möglich.
- Festlegung von Containergrößen zur Laufzeit mit der neuen Funktion SetTableSize. Diese Option wird z. Zt. nicht von allen Leserklassen unterstützt.
- Funktion ResetTable erhält einen zweiten Parameter
- Die Funktionen GetData und SetData der Basisklasse FEDM_Base sind jetzt public. Man kann sie so als zusätzliche globale Funktionen für selbstdefinierte Datencontainer vom Typ FEDM_BYTE_ARRAY verwenden.

V1.02.00

- Die Portierung auf Borland C++Builder ist abgeschlossen.
- Es wurde die Fehlerbehandlung in FEDM_DataBase komplettiert.
- Kleine Fehlerkorrekturen und Ergänzungen
- Neue abstrakte Funktionen in der Basisklasse FEDM_DataBase

V1.00.00

- erste Release-Version