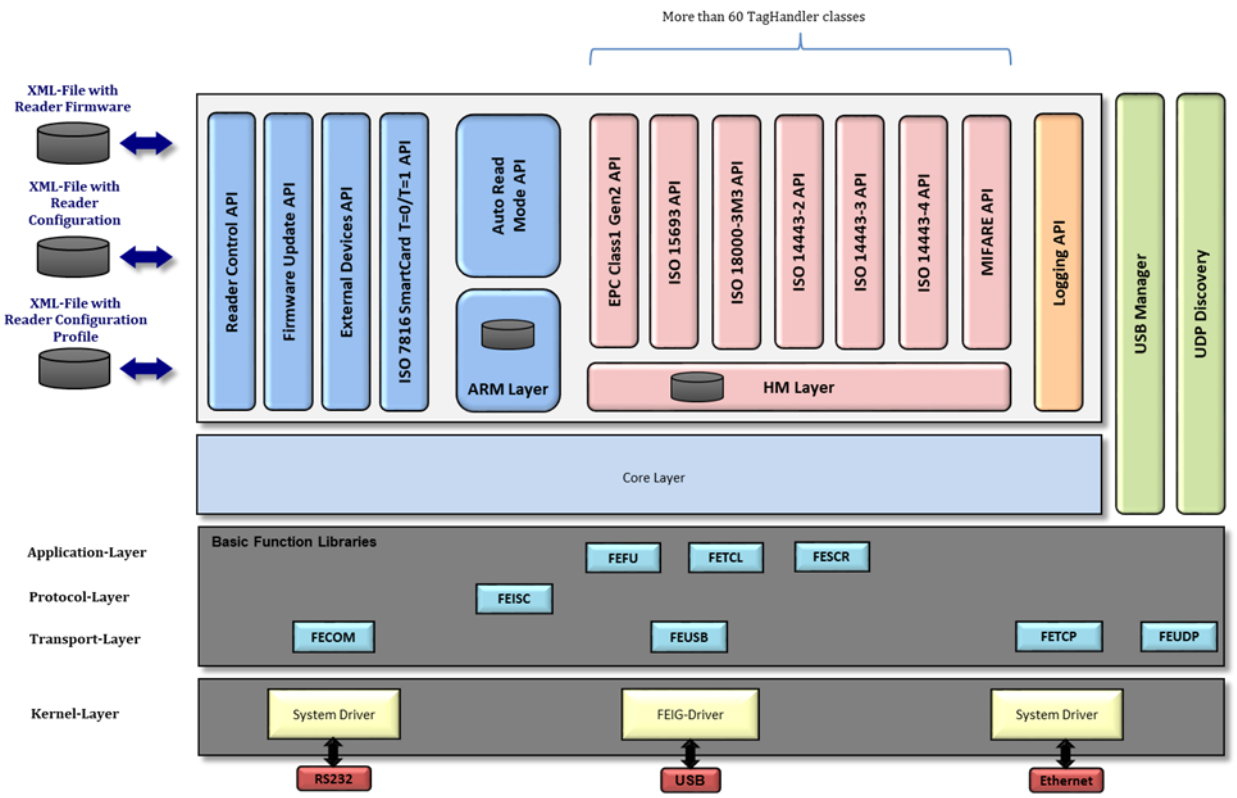
IDENTIFICATION

APPLICATION NOTE

ID ISC.SDK.xxx

Guideline for migration from old to new API for C++ / C# / Java



Note

© Copyright 2018 by

FEIG ELECTRONIC GmbH

Lange Straße 4

D-35781 Weilburg

Germany

eMail: identification-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, giving it to others and the use or communication of the contents thereof, is 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 a FEIG system in cross surroundings.

General Information Regarding this Document

- The following figure formats are used:
  0...9: for decimal figures
  0x00...0xFF: for hexadecimal figures,
  b0...1 for binary figures.
- The hexadecimal value in brackets "[ ]" marks a command.

Content

1. Safety Instructions / Warning - Read Before Start-Up!

- The device may only be used for the intended purpose designed by for the manufacturer.
- The operation manual should be conveniently kept available at all times for each user.
- Unauthorized changes and the use of spare parts and additional devices which have not been sold or recommended by the manufacturer may cause fire, electric shocks or injuries. Such unauthorized measures shall exclude any liability by the manufacturer.
- The liability-prescriptions of the manufacturer in the issue valid at the time of purchase are valid for the device. The manufacturer shall not be held legally responsible for inaccuracies, errors, or omissions in the manual or automatically set parameters for a device or for an incorrect application of a device.
- Repairs may only be executed by the manufacturer.
- Installation, operation, and maintenance procedures should only be carried out by qualified personnel.
- Use of the device and its installation must be in accordance with national legal requirements and local electrical codes.
- When working on devices the valid safety regulations must be observed.

2. Revision History

Revision	*Date*	*Description*
1.0	*2018-07-27*	*First revision*

3. Introduction

FEIG ELECTRONIC GmbH has developed different, hierarchical structured software libraries, assembled in SDKs, to simplify the integration of FEIG readers into customer's applications.

A common attribute of all libraries is the support of all FEIG readers with a uniform Application Programming Interface (API).

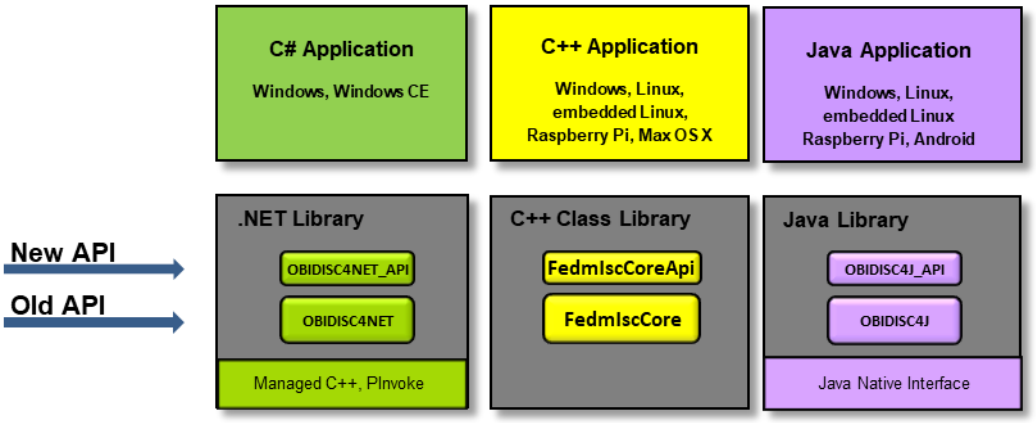
Class libraries for the most popular programming languages C++, C# and Java represent the highest level in the software stack.

This application note contains a guideline for smooth migration from old API to new API.

What is old API?

The first generation of object-oriented class libraries for C++, C# and Java were designed for five, completely different FEIG reader families with different binary communication protocols to unify the high-level handling. In the years from 2000 to ~2015, many new FEIG readers with new features were released, while most of all FEIG reader families, except one, reaches end of life. During these stormy years, the API grows very fast and many classes and constants must be added with the result, that the introduction for new programmers became more and more tricky.

Thanks to the varied feedback from the programmers, we decided to design a new API on top of the old API, which is downgraded to an implementation layer.



The improvements and benefits are:

- New API wraps the old API
- Common high-level API for all FEIG readers
- API is almost identical for C++, C# and Java
- Breakdown the complex reader classes into namespaces and groups of method interfaces
- Arrangement of all constants in namespaces
- API documentation in HTML
- Remove of ballast

The best thing about this design decision is the availability of the old API through the new API. This enables the use of both APIs in one program and therefore a gradual migration of existing projects.

The day will come when the old API will be discontinued. This step is inevitable to ensure the maintainability of the libraries. Because fundamental innovations are implemented only in the new API, it is strongly recommended to start new projects only with the new API and to migrate existing projects that will continue to evolve for many years.

4. Steps for Migration

This chapter collects the most important migration steps. Further information about the new API can be found in the Tutorial (H60810-e-ID-B) and in the HTML Help.

Important note 1: when not other stated, then all references to namespaces, classes and structures inside this guide are in C++ notation. This helps to simplify the descriptions as Java and C# notation is similar.

Important note 2: For reasons of clarity the processing of the return values of the methods is not shown here. Of course it should always be included in applications.

4.1. Namespaces

It is possible to use the namespaces/package names of old and new API together in one project.

	Old API	New API
C++	-	FEDM::Core FEDM::Core::TagHandler FEDM::Core::ReaderCommand FEDM::Core::ReaderConfig FEDM::Core::Const FEDM::Core::ExternalDevices FEDM::Core::Utility
C#	OBID OBID.TagHandler OBID.ReaderCommand OBID.ReaderConfig	FEDM.Core FEDM.Core.TagHandler FEDM.Core.ReaderCommand FEDM.Core.ReaderConfig FEDM.Core.Const FEDM.Core.ExternalDevices FEDM.Core.Utility
Java	de.feig de.feig.TagHandler de.feig.ReaderCommand de.feig.ReaderConfig	de.feig.FEDM.Core de.feig.FEDM.Core.TagHandler de.feig.FEDM.Core.ReaderCommand de.feig.FEDM.Core.ReaderConfig de.feig.FEDM.Core.Const de.feig.FEDM.Core.ExternalDevices de.feig.FEDM.Core.Utility

4.2. Reader Class

The first step for migration is to change the reader class.

	Old API	New API
C++	#include “FedmIscCore.h” FEDM_ISCReaderModule	#include “FedmIscCoreApi.h” FEDM::Core::ReaderModule
C#	OBID.FedmIscReader	FEDM.Core.ReaderModule
Java	de.feig.OBID.FedmIscReader	de.feig.FEDM.Core.ReaderModule

Only little modifications are necessary to change the reader class, when your reader instance is created at runtime:


```
#include "FedmIscCoreApi.h"
using namespace FEDM::Core;


ReaderModule* reader = new ReaderModule();  // reader object from new API

// get pointer of embedded reader object of old API
FEDM_ISCReaderModule* oldReader = reader->GetReaderImpl();


// up from this point, you can use oldReader without any restrictions.
// This means: your code keeps untouched.
```

If your existing project has a static reader instance, then you must change the use of the reader object in the whole project to a dynamic instance.

4.3. Basic Initializations

The second step for migration is to initialize the reader instance.

4.3.1. Setting the Table Size

The integrated tables for Buffered Read Mode (BRM) rsp. Notification Mode (NTF) and Host Mode (HM) are not initialized. Before the initial communication, you must set the table size. The size is selected equal to the maximum number of transponders located in the antenna field to be processed to at the same time.

Please consider that a FEIG reader can process only one working mode. Thus, only the size of the table being used needs to be set.

Old API

```
#include "FedmIscCore.h"


FEDM_ISCReaderModule reader;   // reader instance

// initialize the table for Host Mode
reader.SetTableSize(FEDM_ISC_HM_TABLE, 100);

// or

// initialize the table for Buffered Read Mode or Notification Mode
reader.SetTableSize(FEDM_ISC_BRM_TABLE, 100);
```

New API

```
#include "FedmIscCoreApi.h"
using namespace FEDM::Core;


ReaderModule reader;// reader instance

// initialize the table for Host Mode
reader.IHmTable.SetSize(100);

// or

// initialize the table for Buffered Read Mode or Notification Mode
reader.IBrmTable.SetSize(100);
```

4.3.2. Setting the Reader Type

The reader type must be set in the reader class with one of two options:

1. The call of one of the ConnectXXX methods, which calls internally ReadReaderInfo after a successful connection (recommended).
2. Set of reader type with the method SetReaderType. The constants of all reader types are listed in the namespace FEDM::Core::Const::ReaderType (formerly constants e.g. FEDM_ISC_TYPE_ISCMR102 in the file FEDM_ISC.h)

Old API

```
#include "FedmIscCore.h"
#include "FEDM_ISC.h"


FEDM_ISCReaderModule reader;   // reader instance

// setting of reader type
reader.SetReaderType(FEDM_ISC_TYPE_ISCMR102);
```

New API

```
#include "FedmIscCoreApi.h"
using namespace FEDM::Core;


ReaderModule reader;// reader instance

// setting of reader type
reader.SetReaderType(Const::ReaderType::ID_ISC_MR102);
```

4.4. Establish a Connection to the Reader

The third step for migration is to establish a connection to the reader.

For C++ only: in difference to the old API, the ConnectXXX methods read the reader info internally after a successful connection.

All connection related methods, types and constants are assembled in the nested class IPortGroup, which it's instance is the reader's public member IPort.

Example for a serial connection:

Old API

```
#include "FedmIscCore.h"


FEDM_ISCReaderModule reader;   // reader instance

// initializing the reader instance…

// prepare BusAddress=2
reader.SetBusAddress(2);
// serial connection with port number 1 to reader with default values: Baudrate=38400, Frame=8E1
reader.ConnectCOMM(1);
// detect reader and adjust port settings
reader.FindBaudRate();// blocking call
// read complete reader info to initialize the reader instance
reader.ReadReaderInfo();
```

New API

```
#include "FedmIscCoreApi.h"
using namespace FEDM::Core;


ReaderModule reader;// reader instance

// initializing the reader instance…

// prepare serial settings for connection
// (very useful, if the connection settings of the reader is known)
IPortGroup::SerialPortSettings portSettings(2, 38400, "8E1");
// serial connection with port number 1 to reader with prepared port settings, detection of the reader
(parameter 2) and internal call of ReadReaderInfo.
reader.IPort.ConnectCOMM(1, true, portSettings);
```

4.5. Nested Interfaces in Reader Class

All following steps for migration depends on the application's task.

As the method interface is near to the old API, it is helpful to be informed about the reader's internal nested class interface, which tailors the complex method interface into specific groups. So, if a method of old API is to be migrated, the specific group is first to be identified before the new method name and signature can be found. Some examples in the HTML-Help demonstrate also the use of the methods.

Nested interface	Class name	Description
	ReaderModule	fundamental methods for initialization and reader control
IPort	IPortGroup	interface providing port specific methods like ConnectUSB, etc.
ICmd	ICmdGroup	interface providing reader command specific methods
ICfg	ICfgGroup	interface providing reader configuration specific methods
IInfo	IInfoGroup	interface providing reader info specific methods
ITag	ITagGroup	interface providing transponder specific methods like Inventory and Select
IHmTable	IHmTableGroup	interface providing Host Mode table specific methods. Normally, nested class interface of ITag should be preferred for transponder communication
IBrmTable	IBrmTableGroup	interface providing Buffered Read Mode table specific methods
IAsync	IAsyncGroup	interface providing special methods for asynchronous communication
ILog	ILogGroup	interface providing special methods for logging
IKeyMng	IKeyMngGroup	interface providing special methods for key management
IFwUpd	IFwUpdGroup	interface providing methods for firmware update
IExtDev	IExtDevGroup	interface providing specific methods for external devices (like Gate People Counter or Function Units)