FL3X Config SDK

User Manual

Contact Information

STAR ELECTRONICS GmbH & Co. KG
A Company of the STAR COOPERATION Group
Jahnstraße 86
73037 Goeppingen
Phone: +49 (0)7031 6288-5656
Phone: +49 (0)7031 6288-5330 (Support)

Sales: sales-ee@star-cooperation.com Support: support-ee@star-cooperation.com
www.flex-product.com

Company Data

STAR ELECTRONICS GmbH & Co. KG, registered offices: Goeppingen, register court Ulm, HRA 721096
Partner liable to unlimited extent: STAR ELECTRONICS Verwaltungs-GmbH, registered offices: Goeppingen, register court Ulm, HRB 722565
Represented by the executive board: Rolf Wittig, Henning Lange

Copyright Notice

Trademarks

Any trademarks used in this document are the property of their respective owners.

Disclaimer

The information contained in this document does not affect or change General Terms and Conditions of STAR ELECTRONICS GMBH & CO. KG. STAR ELECTRONICS GMBH & CO. KG does not guarantee the completeness and accuracy of the content of this document and assumes no responsibility for any errors which may appear in this document or due to this document. The content of this document or the associated products are subject to change without notice at any time.

It is currently impossible to develop software that is bug-free in all applications. Therefore, the product is only allowed to be used in the sense of the product use case described herein.

STAR ELECTRONICS GMBH & CO. KG makes no warranty express or implied, as to this document or the information content, materials or products for any particular purpose, nor does STAR ELECTRONICS GMBH & CO. KG assume any liability arising out of the application or use of this product, and disclaims all liabilities, including without limitation resulting damages, as permissible by applicable law.

All operating parameters which are provided in this document can vary in different applications or over time. The herein described product, may solely be used as described in chapter 1.2 “Intended use”.

Any semiconductor devices have an inherent chance of failure. You must protect against injury, damage or loss from such failures by incorporating safety design measures into your facility and equipment such as redundancy, fire protection, and prevention of over-current levels and other abnormal operating conditions. The safety and handling instructions in this document must be followed strictly.

Revision history:

VERSION	DATE	DESCRIPTION
1.5.2.0	09 February 2023	Bugfixes and converter updates, Rename product name to FL3X Config SDK
1.4.3.0	03 June 2022	Bugfixes and converter updates, Database caching and automatic distribution image update
1.3.2.0	20 December 2021	Bugfixes and converter updates
1.2.4.0	09 September 2021	Fix for delivery package
1.2.3.0	24 June 2021	Support for automotive ethernet and SOME/IP added
1.1.0.0	10 December 2020	Support for CAN / CAN FD Transmit
1.0.5.0	13 May 2020	Minor corrections
1.0.4.0	06 April 2020	First official release

Related Hardware / Software Versions

PRODUCT	REFERENCE NO.	VERSION	REMARKS
FL3X Device-S	3-0086-0A01-30		Old product name: FlexDevice-S

PRODUCT	REFERENCE NO.	VERSION	REMARKS
FL3X Device-L	3-0087-0A02-01	Rev 2	Old product name: FlexDevice-L
FL3X Device-L	3-0087-0A03-01	Rev 3	Old product name: FlexDevice-L
FL3X Device-L²	3-0087-0S02-01	Rev 2	Old product name: FlexDevice-L²
FL3X Device-L²	3-0087-0S03-01	Rev 3	Old product name: FlexDevice-L²
FL3X Device-PXIe	3-0094-0B01	Rev 1	Old product name: FlexCard PXIe3
FL3X Device-PCIe	3-0095-0B01	Rev 1	Old product name: FlexCard PXIe3
FL3X Interface-PMC	3-0055-0A01	Rev 1	Old product name: FlexCard PMC-II
FlexCard USB-M Hardware	3-0058-0A01		
fcBase API	3-0009-0K03	S6V9	

Ordering Information

PRODUCT	DESCRIPTION	ORDERING NUMBER
FL3X Config-SDK	FL3X Config SDK main product	3-V0161K01-01
FL3X Config-SDK Maintenance	Maintenance for one year	3-V0161L01-01

Related documents

DOCUMENT	DESCRIPTION	DOCUMENT NUMBER
Usermanual	This document	3-0016-1K01-D11
fcBase API Usermanual	Usermanual for the FlexCard Base API	3-0009-0S01-D03

Contents

Introduction
  Introduction
Getting started
  Databases
  Devices
  Configuration
  Monitoring
  Transmit
  Interpreter
Api Documentation
  FlexConfig.Sdk.Caching
    Extension
    IFileHandler
  FlexConfig.Sdk.Database
    BusType
    ByteOrder
    CanTriggering
    ComputationType
    ContainedCollectionSemanticType
    ContainedTriggerType
    ContainerAcceptMode
    ContainerHeaderType
    DatabaseLoader
    FilterExtension
    FlexRayChannel
    FlexRayTriggering
    IAbsoluteTiming
    IArrayDimension
    IBus
    ICanTriggering
    ICommonDataType
    IComplexDataType
    IComputationMethod

IContainedPduSpecification
IContainerPduSpecification
IDatabase
IDatabaseEntity
IDatabaseLoader
IDataType
IDataTypeDeclaration
IElementLength
IElementPosition
IEthernetConfiguration
IEthernetTriggering
IEvent
IEventGroup
IField
IFilterExtension
IFlexRayTriggering
IFrame
IFrameSpecification
IInterval
ILengthInfo
ILoaderExtension
IMethod
IMultiplexedPduSpecification
IMultiplexerPduSpecification
IntervalType
IPdu
IPduSpecification
IProvidedServiceInstance
IpVersion
IScaleConstraint
IScalingInfo
ISecureCommunicationProperties
ISegmentPosition
IServiceElement
IServiceInstance
IServiceInterface
ISignal

ISignalSpecification
IStandardPduSpecification
ITransportProtocol
ITriggering
LanguageString
LengthInfoType
ScaleConstraintValidity
ServiceElementType
SignalDataType
TransportProtocol
FlexConfig.Sdk.Devices
BusCanErrorType
BusCommunicationControllerState
BusCommunicationErrorType
BusEventEventArgs
BusEventType
BusFlexRayErrorType
BusFlexRayWakeupStatus
BusStatusFlag
CcType
ContainedMultiplexerPdu
ContainedPduType
ContainedStandardPdu
DeviceBusConnectorName
DeviceManager
DeviceType
EthernetMessageType
EthernetMode
FrameDirection
IBusEvent
ICanErrorEvent
ICanFrameEvent
ICommunicationErrorEvent
IDevice
IDeviceBusConnector
IDeviceInfo
IDeviceManager

IEthernetEvent
IEthernetFrameEvent
IEthernetServiceEvent
IFlexRayErrorEvent
IFlexRayFrameEvent
IFlexRayStatusEvent
IFlexRaySyncInformationEvent
IFrameEvent
IMultiplexerSegment
IStatusEvent
ISwitchCodeSignal
ITransmitableContainerPdu
ITransmitableFrame
ITransmitableMultiplexerPdu
ITransmitablePdu
ITransmitableSignal
ITransmitableStaticPdu
ITransmitableSwitchedPdu
MultiplexerSegmentType
FlexConfig.Sdk.Exceptions
FcBaseApiErrorException
NotSupportedDeviceException
FlexConfig.Sdk.Interpreter
DataTypeInterpreterType
IArrayDataInterpreter
ICommonDataTypeInterpreter
IComplexDataTypeInterpreter
IDataTypeInterpreter
IFrameInterpreter
IPduInterpreter
ISignalInterpreter
ISignalValue
Version Notes
Release Notes
Version compatibility
Open Issues
License

License

FL3X Config SDK

Intended User Group

This product may only be used by expert technicians and/or engineers who are qualified and familiar with electronic components and systems. Each person involved with setup or operation of the product must

- be a qualified technician or engineer
- strictly adhere to this manual
- receive a briefing by an authorized person

NOTICE

If you are unsure of how to use the product as intended or have any questions about the use of the product, please discontinue use of the product immediately and contact the STAR ELECTRONICS GmbH & Co. KG Support.

⚠WARNING

The product may only be used by expert technicians and/or engineers who are qualified and familiar with electronic components and systems!
The use of the product by non-professionals is not permitted and strictly forbidden!

Intended use

The FL3X Config SDK is a software API specially designed for testing purposes in connection with STAR ELECTRONICS GmbH & Co. KG's hardware devices. It was developed to configure and control STAR ELECTRONICS GmbH & Co. KG hardware devices (e.g. the FL3X Device product family) exclusively. It is NOT permitted to use the software with or in connection with any other hardware devices. For this intended use, the FL3X Config SDK offers the following options:

- Support for different automotive databases
- Configuration of STAR ELECTRONICS GmbH & Co. hardware
- Control the runtime behavior of STAR ELECTRONICS GmbH & Co. hardware
- Interpretation of data traffic (e.g. Use Case "Analyzing")

Any deviation from the intended use is only permitted with specific **prior written approval** of STAR ELECTRONICS GmbH & Co. KG.

⚠WARNING

Any use of the software on or in connection with a STAR ELECTRONICS GmbH & Co. KG hardware device product outside a fully controlled testing and/or laboratory environment may result in death or serious injury due to unpredictable behavior of a vehicle and/or potentially missing, deactivated or malfunctioning safety devices on a vehicle!
This software allows - only for testing purposes - to change the communication behavior of the STAR ELECTRONICS GmbH & Co. KG hardware device product.

⚠WARNING

The FL3X Config SDK may be used to communicate with networked electronic systems. E.g. FlexRay, CAN or Ethernet.
Any use of the product outside a fully controlled testing and/or laboratory environment may result in death or serious injury due to unpredictable behavior of a vehicle and/or potentially missing, deactivated, or malfunctioning safety devices on a vehicle!
The user is responsible to ensure the safety of the entire system. This includes amongst other things a safety shutdown.

NOTICE

The device is not a calibrated measurement device. STAR ELECTRONICS GmbH & Co. KG accepts no liability whatsoever for the correctness of any measurement results.

⚠WARNING

The FL3X Config SDK is NOT designed, intended, or authorized and may NOT be used for or in connection with the following purposes and/or devices:

- use as part of medical systems
- life support applications
- aviation, space, nuclear, or military applications
- any other purposes / devices deviating from the intended use of the product specified by STAR ELECTRONICS GmbH & Co. KG.

⚠WARNING

The product may only be used by expert technicians and/or engineers who are qualified and familiar with electronic components and systems!
The use of the product by non-professionals is not permitted and strictly forbidden!

Safety and Handling Instructions

Please read the instructions for use carefully. To protect the device or the application against damage, or to avoid personal injury the User Manual must be handled as described herein.

Changes or modifications of the User Manual are not allowed for safety and warranty reasons!

STAR ELECTRONICS GMBH & CO. KG is not liable for any damages arising from non-observance of the product information.

Follow the
a) specific safety and handling instructions placed at dedicated document positions
b) general safety and handling instructions below:

⚠WARNING

FL3X Config SDK can be used to interfere with networked electronic systems. By connected measurement hardware like the FL3X Device-L, FL3X Config SDK can be used to transmit messages via FlexRay, CAN or Ethernet busses.
If transmitted messages are received by real electronic control units, e.g. within a test car, these messages could result in an unpredictable behavior or a failure of the electronic control unit. This may result in serious injury of persons or material damage!
Only qualified and briefed persons may use FL3X Config SDK!
Only transmit messages, where the expected behavior of the receiver is known.

Hardware and software requirements

Currently the supported operating system is Microsoft Windows 10. No special hardware requirements running the FL3X Config SDK exists.

FL3X Device requirements

The following requirements and restrictions exists if you are using FL3X Config SDK with a FL3X Device.

Device Discovery

For FL3X Device family the device discovery uses currently ICPM ping with raw sockets which requires that the process executing FL3X Config SDK needs administrator privileges otherwise no FL3X Devices will be found.

If the ip address is known, the user can provide a hint via SetGlobalConfig(String, String) for the enumeration method, so that no administrator privileges are needed.

```
public class Program
{
   private static void Main(string[] args)
   {
        // IP Address of the known device
       var knownDeviceIp = "192.168.1.15"
       DeviceManager.Instance.SetGlobalConfig("hwComAddress", knownDeviceIp);

       var deviceInfos = deviceManager.GetDevices();
```

Required TCP/UDP Ports

Using the FL3X Devices with the FL3X Config SDK on a network protected by a Windows Firewall, the firewall must be configured to permit FL3X Config SDK to access the FL3X Device network resources.

The following ports must be open on the computer running the FL3X Config SDK:

PORTS	INCOMING/OUTGOING	PROTOCOL	DESCRIPTION
1500	Outgoing	TCP	Used for HW-Com normal protocol operation
15300	Incoming	UDP	Used for HW-Com Streaming. In case the port is already used, the next free port after 15300 will be used.
15300	Outgoing	UDP	Used for HW-Com frame transmissions

Frame Transmission

Please note that buses, which have already been configured with a FL3X Config RBS project, will not be able to transmit messages. When transmitting messages, use a bus on your hardware, which has no database configured via FL3X Config RBS.

If frame transmission fails on a FL3X Device a ICommunicationErrorEvent is added to the list of events which are received via OnReceive.

Connection lost event

If the network connection to the FL3X Device is lost during measurement, the OnConnectionLost event will be raised.

Multiple applications - Device locking

Normally the API will prevent that multiple applications will access the same device. However for the FL3X Device it cannot be guaranteed that no other application will get access to the API functionality at the same time as the lock is based on the network

connection (IP address) of the PC.

If multiple FL3X Config SDK applications are running on the PC at the same time, the user must ensure that the device is only accessed by the API from one application. Otherwise undefined behavior can occur.

Automatic distribution image update

With each release the folder `images` will contain the newest distribution image which can be used to automatically update the device image.

To use the automatic update functionality the following must be considered. An automatic update is done when

- the files from the deployment folder `images` are in the current working directory as the updater uses this directory to scan for the `s19` files. To use a different folder, the user can specify the path via the method SetGlobalConfig(String, String) e.g.:

```
DeviceManager.Instance.SetGlobalConfig("DistributionImageFolder", @"c:\MyApplication\files");
```

- the device type is a FL3X Device and is running a distribution image. An image with a running RBS will not be updated.
- the update will be done during opening the device, which means that this method call will take longer if the device is updated.

Supported Hardware

PRODUCT	REFERENCE NO.	VERSION	REMARKS
FL3X Device-S	3-0086-0A01-30		Old product name: FlexDevice-S
FL3X Device-L	3-0087-0A02-01	Rev 2	Old product name: FlexDevice-L
FL3X Device-L	3-0087-0A03-01	Rev 3	Old product name: FlexDevice-L
FL3X Device-L²	3-0087-0S02-01	Rev 2	Old product name: FlexDevice-L²
FL3X Device-L²	3-0087-0S03-01	Rev 3	Old product name: FlexDevice-L²
FL3X Device-PXIe	3-0094-0B01	Rev 1	Old product name: FlexCard PXIe3
FL3X Device-PCIe	3-0095-0B01	Rev 1	Old product name: FlexCard PXIe3
FL3X Interface-PMC	3-0055-0A01	Rev 1	Old product name: FlexCard PMC-II
FlexCard USB-M Hardware	3-0058-0A01		

Installation

There is no special installation requirements apart from the installation of the Wibu Codemeter software. The library files from `bin\x86` and/or `bin\x64` can be simply copied to desired location.

Wibu Codemeter

To run any software with the FL3X Config SDK the Wibu protection system must be installed. The folder `bin\wibu` contains the required software package which can be used to install the Wibu CodeMeter software.

Development

FL3X Config SDK is based on .NET Standard 2.0, so any .NET runtime which supports .NET Standard 2.0 can be used.

FL3X Config SDK supports x86 and x64 platforms. The following files are platform dependent:

- FlexConfig.Sdk.dll
- Ebel.Hardware.FlexCard.dll
- fcBase.dll

All files which are required can be found in the folders `bin\x86` or `bin\x64`.

Software protection

The FlexConfig.Sdk.dll is secured with the Wibu protection system. Any program which uses the FlexConfig.Sdk.dll requires a dongle with a valid FL3X Config SDK license.
Without a Dongle, no software using FL3X Config SDK can be executed. During runtime the license is also checked. If the dongle is removed during runtime the FL3X Config SDK will shutdown the whole software when calling specific methods from FL3X Config SDK.

Deployment

The minimum requirement for the deployment of your software consists of the CodeMeter License Server and the FL3X Config SDK Library files (`bin\x86` and/or `bin\x64`)

For the Codemeter deployment there exists installation packages for Windows. As alternative the packages can be downloaded from the user area of the Wibu-Systems website (http://www.wibu.com/en/downloads-user-software.html).

Currently the used software protection system supports only Microsoft Windows 10 system.

Introduction

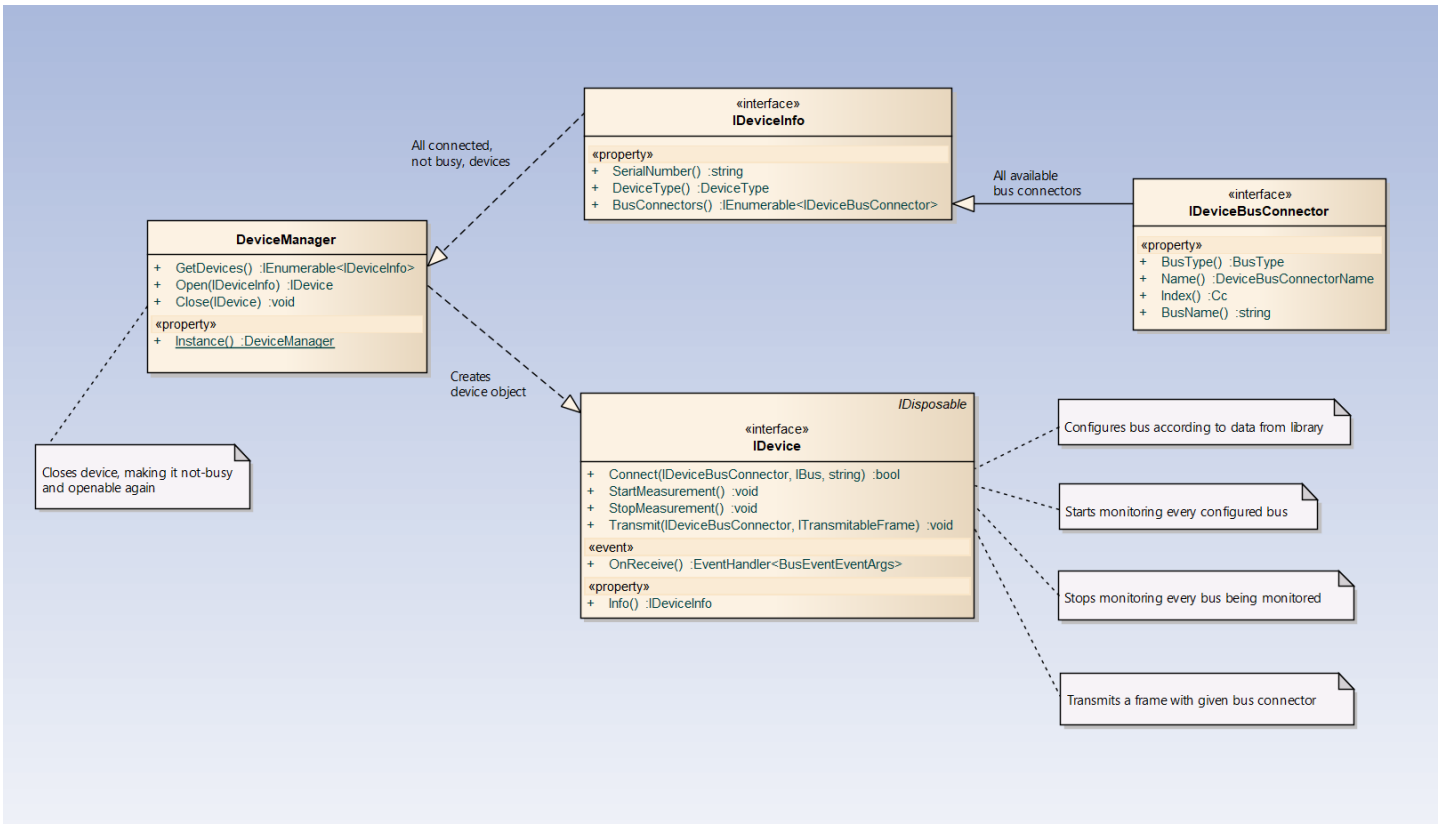
FL3X Config SDK is used to enumerate, open, connect and monitor FlexCards or FlexDevices, transmit frames and interpret the bus information as user friendly as possible.

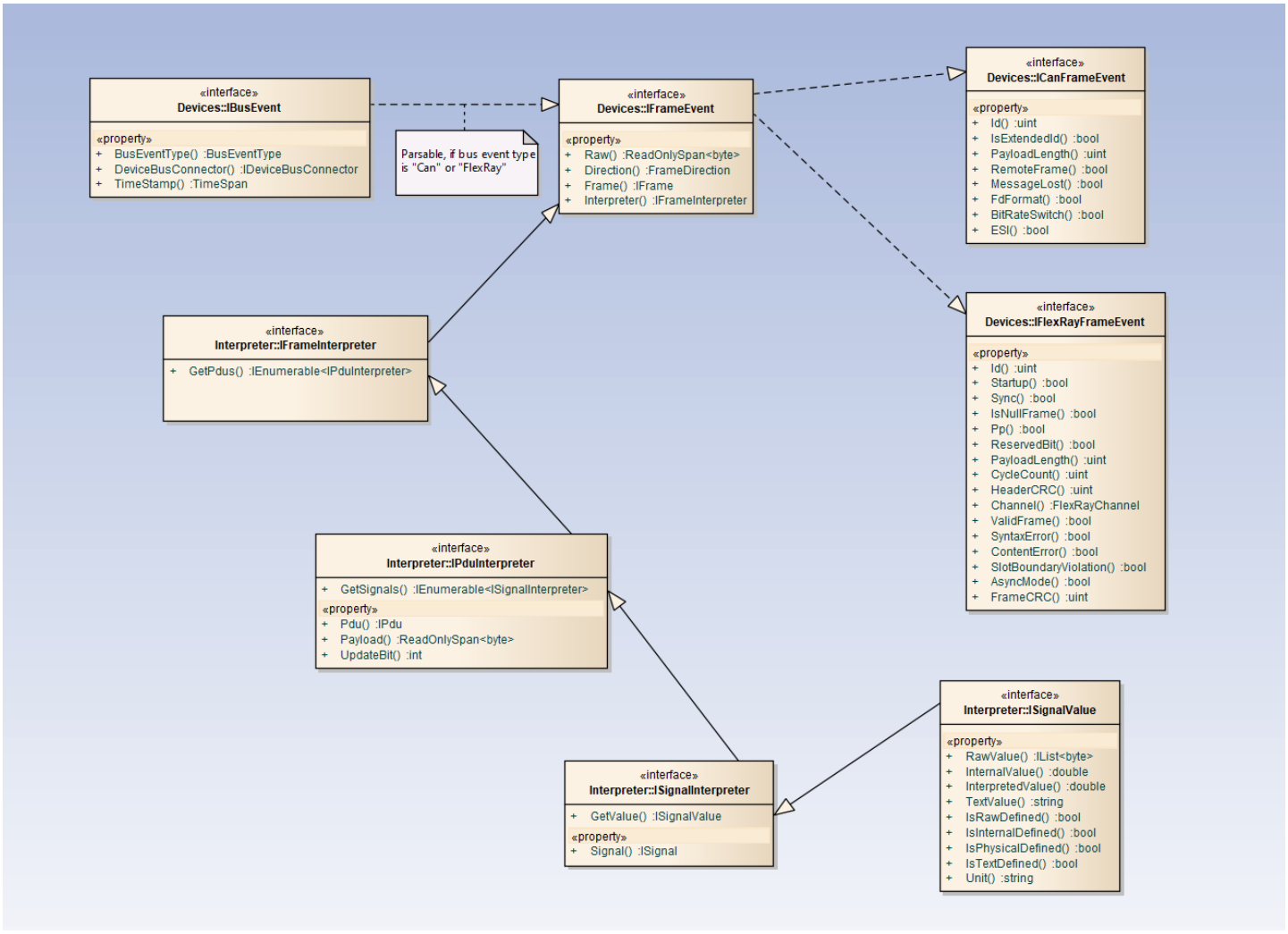
With a small amount of code, your FlexCard or FlexDevice can be configured with automotive database like AUTOSAR, FIBEX or DBC to receive and interpret frames and signals from bus systems like FlexRay, CAN or CAN-FD.

CAN and CAN-FD frames from prior mentioned database types can be used to create transmitable frames, which are editable and transmitable.

The following articles will give you some information about the basic workflow and guide you through the easiest way to setup your FlexCard or FlexDevice almost completely automatically.

- Loading database
- Accessing Hardware
- Configuration
- Monitoring
- Interpreter





Example

The following code shows an example on how a project using the FL3X Config SDK may look like:

```
public class Program
{
    private static void Main(string[] args)
    {
        // Loading Database
        var databaseLoader = DatabaseLoader.Instance;
        var database = databaseLoader.Load(" [Path to database] ");

        // Device Open
        var deviceManager = DeviceManager.Instance;
        var deviceInfos = deviceManager.GetDevices();
        var device = deviceManager.Open(deviceInfos.First());

        // CC Configuration
        var canfdConnector = device.Info.BusConnectors.FirstOrDefault(x =>
            x.BusType == BusType.CANFD && x.Name == DeviceBusConnectorName.Connector4A);

        var canfdBus = database.Buses.FirstOrDefault(x => x.BusType == BusType.CANFD);

        device.Connect(canfdConnector, canfdBus);

        // Get transmitable frame
        var frame = canfdBus.Frames.FirstOrDefault(x => x.Id == " [Frames_ID_Defined_In_Database] ");
        var transmitableFrame = frame.GetTransmitableFrame();

        // Set Pdu payload
        var pdu = transmitableFrame.Pdus.FirstOrDefault();
        var pduPayload = new byte[] { 128, 64, 0, };
        pdu.SetPayload(pduPayload);
        pdu[2] = (byte)15;

        // Set Signal interpreted value
        var signal = pdu.Signals.FirstOrDefault();
        signal.InterpretedValue = 4;

        // CC Monitoring
        device.OnReceive += DeviceOnReceive;
        device.StartMeasurement();

        // Send frame
        device.Transmit(canfdConnector, transmitableFrame);

        System.Console.WriteLine("Press a key to stop measurement...");
        System.Console.ReadKey();

        device.StopMeasurement();
    }


    // Message Interpreter
    private static void DeviceOnReceive(object sender, BusEventEventArgs e)
    {
        foreach (var busEvent in e.Events)
        {
            if (busEvent is IFrameEvent frameEvent)
            {
                foreach (var pduInterpreter in frameEvent.Interpreter.GetPdus())
                {
                    foreach (var signal in pduInterpreter.GetSignals())
                    {
                        var signalValue = signal.GetValue();

                        var intepretedValue = string.IsNullOrEmpty(signalValue.Unit)
                            ? signalValue.InterpretedValue.ToString(
                                CultureInfo.CurrentCulture)
```

```
                        : signalValue.InterpretedValue.ToString(
                            CultureInfo.CurrentCulture) +
                        signalValue.Unit;

                    var value = signalValue.IsTextDefined
                        ? FormattableString.Invariant(
                            $"{intepretedValue} [{signalValue.TextValue}]")
                        : intepretedValue;

                    System.Console.WriteLine($"{signal.Signal.ShortName}: {value}");
                }
            }
        }
    }
}
```

Database

The DatabaseLoader class is used to load automotive database with contains the bus configuration and the information to interpret the frames on the bus.

Getting an instance

To access methods from the DatabaseLoader, an instance of its class is needed. An instance is already created while runtime, which can be get retrieved via the static Instance method.

Example:

```
var loader = FlexConfig.Sdk.Database.DatabaseLoader.Instance;
```

Loading a database

This method loads the automotive database from the given path. The returned IDatabase object can be used to configure the hardware and to inspect which buses, frames, pdus and signals are defined.

The supported database formats depends on the available plugins. Currently different AUTOSAR, FIBEX and DBC versions are supported.

Example:

```
var database = loader.Load("path_to_database\Fibex_Example.xml");
```

See ref: DatabaseLoader, IDatabase

Using caching to improve load times

To improve the loading times of the database it is possible to enable database caching. To use the caching feature you must reference the `FlexConfig.Sdk.Caching.dll` and enable it via AddCaching(IDatabaseLoader, String) extension method. The calls with Load(String) **must not** be changed. The Load(String) checks if the given file is in the cache (based on md5 hash of the file content). If not, the file will be written to caching folder. The next time the Load(String) will load the optimized version from the cache.

Example:

```
var loader = FlexConfig.Sdk.Database.DatabaseLoader.Instance;

var cachingFolder =
Path.Combine(System.Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData,
"FlexConfig.Sdk");
loader.AddCaching(cachingFolder);
```

See ref: Extension, IDatabase

Encoding of the database file

Ensure that the file has the correct encoding based on your system setup. If not it may happen that some strings are not parsed properly which may results that e.g. a unit of signal is not recognized correctly. The loading mechanism uses the Default encoding which depends on the runtime your are using. The .NET Framework uses always an encoding based on the system's active code page. On .NET Core application the Default encoding is UTF8Encoding.

Accessing the database information

The IDatabase interface contains the information from the automotive database required to configure the devices and to interpret the frames on the buses.

Properties

- To access the bus information
- To get the frames defined in the database

Bus information

The IBus interface contains the information about one bus e.g. FlexRay or CAN-FD. This information includes the type of bus, the configuration which is needed to configure the FlexCard or FL3X Device hardware and which frames are transmitted on the bus.

Devices

With a IDeviceManager object all connected and available devices can be found and accessed. For FL3X Device family please consider the following requirements and restrictions.

Getting an instance

To access methods from the IDeviceManager, an instance of its class is needed. An instance is already created while runtime, which can be get retrieved via the static Instance method.

Example:

```
var deviceManager = FlexConfig.Sdk.Devices.DeviceManager.Instance;
```

Enumerating

This method searches for all connected and available devices and returns them as IDeviceInfo objects. A busy (not available) device would be a device which was already being opened by the device manager or by another application. To make a device not busy, it has to be closed first and searched for again.

Example:

```
var deviceInfos = deviceManager.GetDevices();
```

See ref: IDeviceInfo

Opening a device

This method opens a device according to the given IDeviceInfo object, marks this device exclusive used by the FL3X Config SDK. To access functionalities of the device, the returned IDevice object is used for.

Example:

```
var device = deviceManager.Open(deviceInfo);
```

See ref: IDeviceInfo, IDevice

Closing a device

This method closes a device according to the given IDevice object, making it available for others again.

Example:

```
deviceManager.Close(device);
```

See ref: IDevice

Information about a device

With the IDeviceInfo class, information about the according device can be get.

Each device has its own serial number to identify the device. The device type property contains the type of hardware, e.g. FlexCard PMC2 or FL3X Device S.

An important role in configuring the hardware has the property BusConnectors. The Bus connectors describes the different hardware connectors available on the device. This includes which bus type the hardware supports on this connector, the name to identify on which physical connector the bus is available and type of communication controller.

Configuration

With the IDevice interface, buses can be connected and monitored. While connecting a bus, the device hardware will be configured automatically according to its IBus configuration setting.

Configure a device bus connector

The Connect(IDeviceBusConnector, IBus, String) connects a database bus to a device physical bus which handles all configuration needed so that the device can receive frames from the bus.

Remark: *Once the device bus connector is connected to the database bus, it will be set as configured, which is necessary to start monitoring it.*

To connect and configure with this method, two parameters are needed (the third one is optional):

1. the physical bus connector which should be used from the BusConnectors,
2. the database bus
3. and optional a user defined bus name

Example:

```
// Get the body can information from the database
var bodyCANBus = database.Buses.FirstOrDefault(x => x.ShortName == "Body-CAN" && x.BusType == BusType.CAN);

// The CAN is/should be connected to the devices Connector 4A.
var physicalBusConnector4A  = device.Info.BusConnectors.FirstOrDefault(x =>
            x.BusType == BusType.CAN && x.Name == DeviceBusConnectorName.Connector4A)

// Connect the physical bus to the bus from the database
var connected = device.Connect(physicalBusConnector4A, bodyCANBus, "Body-CAN");
```

Configure a device bus connector for Ethernet/BroadR-Reach

If an Ethernet connector is being configured, then the EthernetMode and the needed VLAN tags must be defined as this is not extracted from the database.

Example:

```
...
var ethernetConnector = device.Info.BusConnectors.FirstOrDefault(x =>
                    x.BusType == BusType.Ethernet && x.Name == DeviceBusConnectorName.Connector1C);
var ethBus = database.Buses.FirstOrDefault(x => x.BusType == BusType.Ethernet);

// Configure the connector as BroadR-Reach Slave 100BaseT1 and the Vlans 5 and 6
device.SetEthernetConfiguration(ethernetConnector, EthernetMode.Mode100BaseT1Slave, new short[1] { 5,6 });
device.Connect(ethernetConnector, ethBus);
```

See references: IDevice, IDeviceInfo, IDeviceBusConnector, IBus EthernetMode

Monitoring

Start the measurement

To start the measurement the method StartMeasurement() must be called. All buses which have been connected via the Connect method will be monitored and are able to receive bus information and transmit new frames. To get the bus events/frames the user has to have registered a handler for the OnReceive event.

Example:

```
device.OnReceive += DeviceOnReceive;
device.StartMeasurement();
```

OnReceive handler

To be notified of new events, the user has to register a handler for the OnReceive event.

The method gets a list of events being received and is able to use interpreters from the message to get:

- Pdus and signals for CAN/FD, FlexRay and Ethernet UDP Frames
- Data types for SOME/IP Service elements (event, field, methode)

The following example shows how to interpret signals/datatypes from a message and write its short name and interpreted value to the console:

```
private static void DeviceOnReceive(object sender, BusEventEventArgs e)
{
    foreach (var busEvent in e.Events)
    {
        if (busEvent is IFrameEvent frameEvent)
        {
            foreach (var pduInterpreter in frameEvent.Interpreter.GetPdus())
            {
                foreach (var signal in pduInterpreter.GetSignals())
                {
                    var signalValue = signal.GetValue();

                    var interpretedValue = string.IsNullOrEmpty(signalValue.Unit)
                        ? signalValue.InterpretedValue.ToString(CultureInfo.CurrentCulture)
                        : signalValue.InterpretedValue.ToString(CultureInfo.CurrentCulture) +
                        signalValue.Unit;
                    var value = signalValue.IsTextDefined
                        ? FormattableString.Invariant(
                            $"{interpretedValue} [{signalValue.TextValue}]")
                        : interpretedValue;

                    System.Console.WriteLine($"{signal.Signal.ShortName}: {value}");
                }
            }
        }
        else if (busEvent is IEthernetServiceEvent serviceEvent && serviceEvent.ServiceElementName ==
"MyEvent")
        {
            if (serviceEvent.Interpreters.FirstOrDefault() is IComplexDataTypeInterpreter complexInterpreter)
            {
                var interpreter = complexInterpreter.Interpreters.FirstOrDefault(m => m.ShortName == "C");

                if (interpreter is ICommonDataTypeInterpreter commonDataTypeInterpreter)
                {
                    Console.WriteLine(commonDataTypeInterpreter.GetValue().InterpretedValue);
                }
            }
        }
    }
}
```

StopMeasurement

This method stops monitoring the buses.

Example:

```
device.StopMeasurement();
```

See ref: IDevice

Transmit

Please note that the current release only supports CAN and CAN-FD frame transmissions.

Create a frame for transmission

Select the frame from the bus Frames collection. With the IFrame object a ITransmitableFrame object can be created.

```
var frame = canBus.Frames.FirstOrDefault(x => x.ShortName == "ABC_STAT1");
var transmitableFrame = frame.GetTransmitableFrame();
```

In this case the first triggering will be used. To select a different triggering use the GetTransmitableFrame(ITriggering) method and specify the triggering to be used.

See ref: IFrame, ITransmitableFrame

The frame

Depending on the frame specification the frame contains standard pdus, container pdus and multiplexer pdus elements.

If one or more of the mentioned pdu types are not defined the corresponding properties will return an empty list.

Read and write access to the frame payload is available via

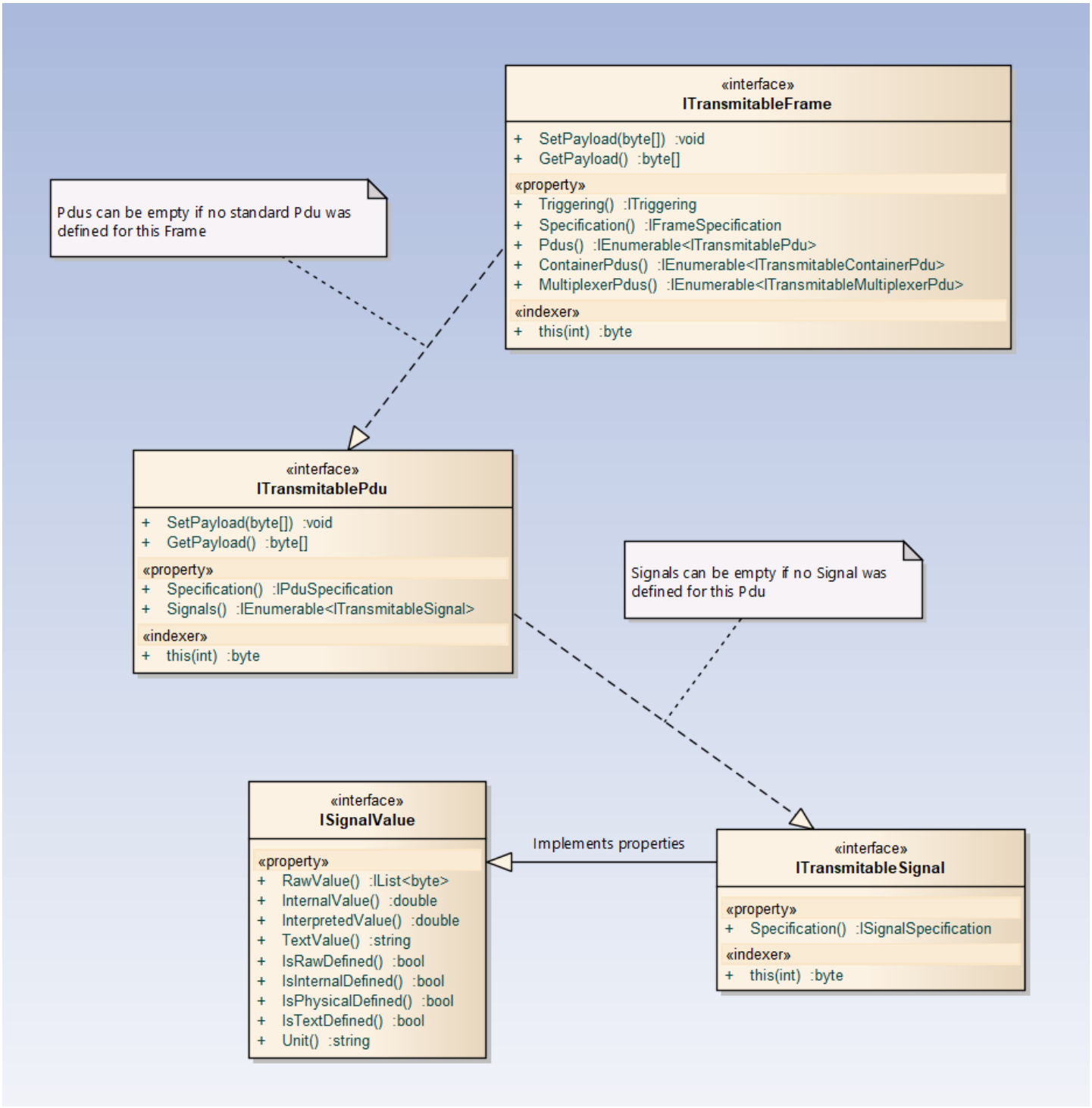
- Item[Int32]
- SetPayload(Byte[])
- GetPayload()

Example:

```
// Frame bit length is 64 bit => Payload size is 8 Bytes ...
transmitableFrame.SetPayload(new byte[] { 1, 2, 3, 4, 5, 6, 7, 8, });
transmitableFrame[0] = 9;
transmitableFrame[7] = 10;
```

See ref: ITransmitableFrame, ITransmitablePdu, ITransmitableContainerPdu, ITransmitableMultiplexerPdu

Standard pdus



A Standard pdu contains signals which can be used to manipulate the payload based on the signal information. The raw payload for a pdu can be accessed similar to the ITransmitableFrame.

```
var standardPdu = transmitableFrame.Pdus.FirstOrDefault(x => x.ShortName == "PDU_STAT1");
var signal = standardPdu.Signals.FirstOrDefault();

 // PDU Bit length is 32 bit
standardPdu.SetPayload(new byte[] { 1, 2, 3, 4, });
standardPdu[0] = 5;
standardPdu[3] = 6;
```

See ref: ITransmitablePdu, ITransmitableSignal

Signals

Manipulating the signal value can be done in several ways.

Raw value

Setting the raw value can be done via indexer or by setting a new a new byte array. Changes to the raw value will directly reflect to the physical value and text value (if specified).

```
// Signal bit length is 12 bit
signal.RawValue = new byte[] { 255, 15, };
signal[0] = 127;
signal[1] = 7;
```

Physical value

Setting the physical value can be done via InterpretedValue property. Changes to the physical value will directly reflect to the raw value and text value (if specified).

```
signal.InterpretedValue = 2;
```

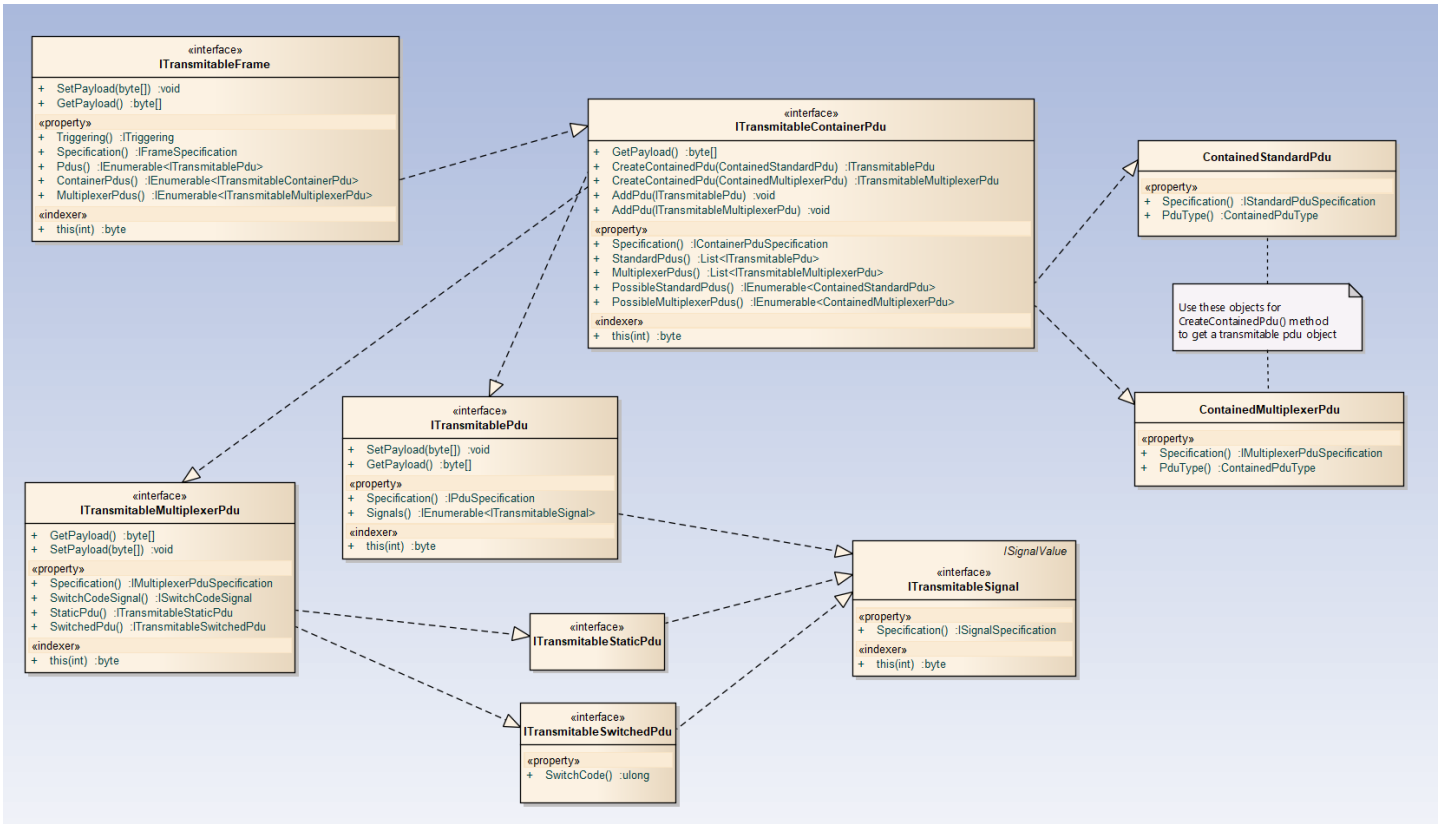
Textual value

If the signal specifies a textual representation, the value can also be changed with a new text value. Changes to the textual value will directly reflect to the raw and physical value.

```
signal.TextValue = "Off";
```

See ref: ITransmitableSignal

Container pdus



Container pdus include contained pdu information about standard pdus and/or multiplexer pdus.

Remark: *The container pdu only knows about the contained pdu information, which were defined in the related frame. While being set to "Accept-all", it is possible to add contained pdu objects, which were created in another frame object.*

If no standard / multiplexer pdu was defined for this container pdu, the corresponding properties will return an empty list.

In order to add a contained pdu, it must be created with the container pdu and the contained pdu information.

```
var standardPduSpec = containerPdu.PossibleStandardPdus.FirstOrDefault();
var containedStandardPdu1 = containerPdu.CreateContainedPdu(standardPduSpec);
```

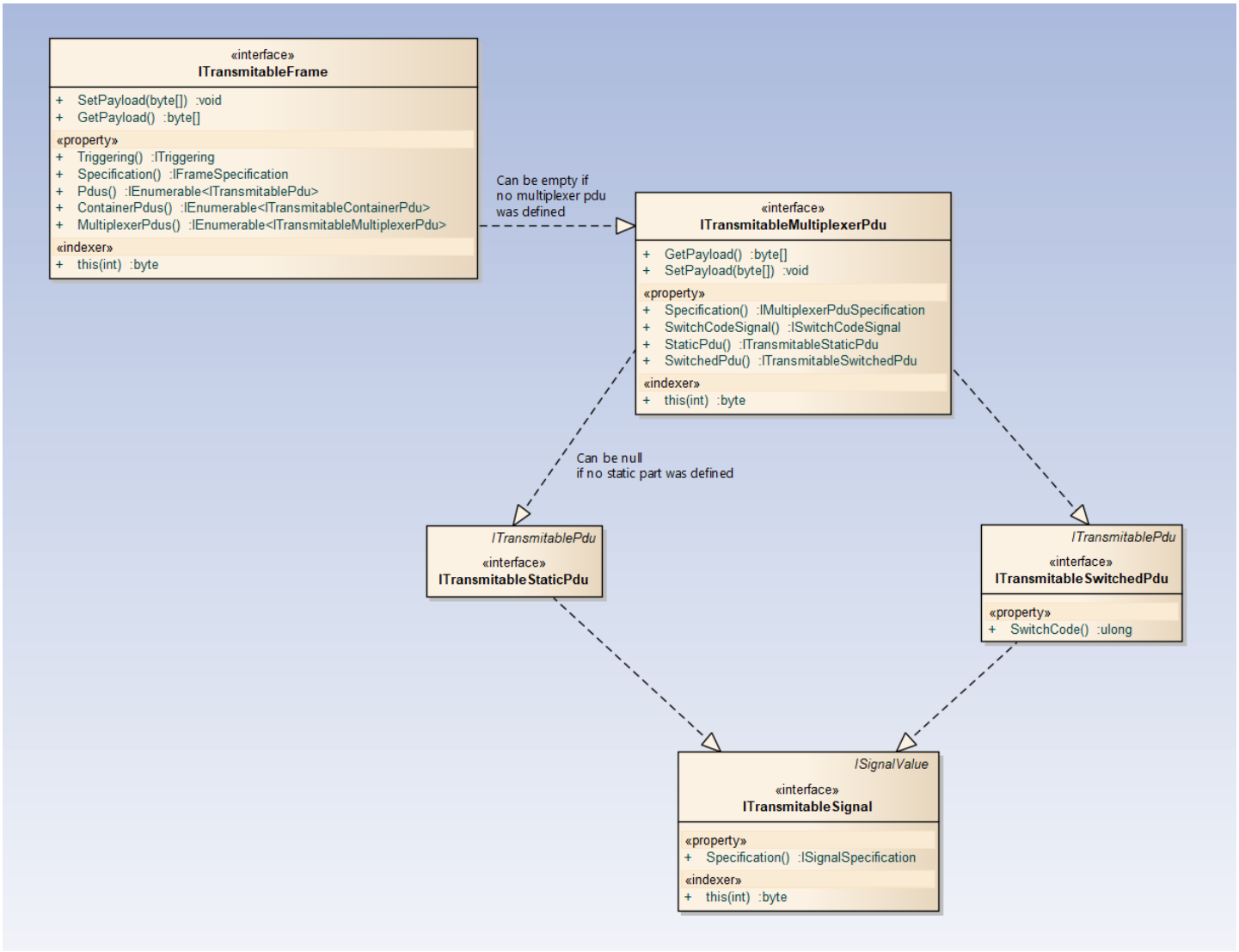
Once created, the contained pdu can be used just like standard / multiplexer pdus.

Remark: *Adding the same contained pdu object (pdu2) to the container pdu, while it has the collection-type "Last-is-Best" and the container pdu already contains the pdu (pdu1), replaces the previous contained pdu object (pdu1). Modifying the replaced pdu object will throw an exception.*

```
var standardPduSpec = containerPdu.PossibleStandardPdus.FirstOrDefault();

var pdu1 = containerPdu.CreateContainedPdu(standardPduSpec); // Last-is-Best

// Do something with contained pdus
containerPdu.AddPdu(pdu1);

var pdu2 = containerPdu.CreateContainedPdu(standardPduSpec); // Last-is-Best

containerPdu.AddPdu(pdu2); // Replaces 'containedStandardPdu1'

pdu2[0] = 254;

// Accessing the replaced pdu will throw a exception
pdu1[0] = 254; // Throws exception.
```

See ref: ITransmitableContainerPdu, ITransmitablePdu, ContainedStandardPdu, ITransmitableMultiplexerPdu, ContainedMultiplexerPdu

Multiplexer pdus



Multiplexer pdus contain a switch code signal, a switched pdu from the switched part, an optional static pdu from the static part and a changeable payload.

The static pdu can be used just like a standard pdu.

The switched pdu is the same with the only exception being the switch code property.

To change the switched pdu, the switch code signal value can be changed with a new switch code (always of type ulong) or a new raw switch code via indexer or as a byte array. It is also possible to change the switched pdu with a new payload for the multiplexer pdu itself, as the multiplexer pdu payload contains the raw switch code.

The first switch code, which is the switch code value of the first defined switched pdu according to the loaded database, will be set in the switch code signal and thus sets the switched pdu. Hence the multiplexer pdu payload will contain, next to the default value 255 / FF, different values according to the bit position and -length of the switch code signal (except the first switch code defined is 255 / FF).

Example:

```
var multiplexerPduBitLength = multiplexerPdu.Specification.BitLength; // For example: 64 bit , 2 switched pdus with switch code 1 and 2
var staticPdu = multiplexerPdu.StaticPdu;
var currentSwitchedPdu = multiplexerPdu.SwitchedPdu; // switch code 1
var switchCodeSignal = multiplexerPdu.SwitchCodeSignal; // For example: bit position = 0 , bit length = 8 bit

multiplexerPdu.SetPayload(new byte[] { 2, 255, 255, 255, 255, 255, 255, 255, });
var currentSwitchedPdu = multiplexerPdu.SwitchedPdu; // switch code 2

switchCodeSignal.SwitchCode = 1;
var currentSwitchedPdu = multiplexerPdu.SwitchedPdu; // switch code 1

switchCodeSignal.RawSwitchCode = new byte[] { 2, };
var currentSwitchedPdu = multiplexerPdu.SwitchedPdu; // switch code 2

switchCodeSignal.RawSwitchCode[0] = 1;
var currentSwitchedPdu = multiplexerPdu.SwitchedPdu; // switch code 1
```

See ref: ITransmitableMultiplexerPdu, ISwitchCodeSignal, ITransmitableSwitchedPdu, ITransmitableStaticPdu

Transmit a frame

In order to transmit a frame, the bus connector configured with the matching database of the frame is needed, while being in monitoring mode.

Via the transmit method of the device object, the frame will be transmitted.

Example:

```
device.Transmit(deviceBusConnector, transmitableFrame);
```

See ref: IDevice, IDeviceBusConnector, ITransmitableFrame

Interpreter

Events from the connected buses are received via OnReceive event. The BusEventEventArgs contains the bus events which can be IFrameEvent or IStatusEvent. A IFrameEvent indicates a valid frame with pdu and signals which can be interpreted with the use of IFrameInterpreter object. The IStatusEvent contains status or error information from the buses.

Example:

```
private static void DeviceOnMessageReceive(object sender, ReceivedMessageEventArgs e)
{
    foreach (var busEvent in e.Events)
    {
        if (busEvent is IFrameEvent frameEvent)
        {
            // Interpret frame
        }
        else if (busEvent is IStatusEvent statusEvent)
        {
            // Handle status event.
        }
    }
}
```

The BusEvent interface.

The IBusEvent interface contains all information about a received event.

Each IBusEvent contains a property BusEventType which can be used to identify the type of event.

The following table shows to which a bus event object can be casted to:

BUS EVENT TYPE	TYPE
Can	ICanFrameEvent
CanError	ICanErrorEvent
FlexRay	IFlexRayFrameEvent
FlexRayStatus	IFlexRayStatusEvent
FlexRayError	IFlexRayErrorEvent
FlexRaySyncInformation	IFlexRaySyncInformationEvent
EthernetFrame	IEthernetFrameEvent
EthernetService	IEthernetServiceEvent

The DeviceBusConnector contains the bus connector on which the event was received or generated.

The TimeStamp marks the time the event was received or generated relative to the measurement start.

The FrameEvent interface

The IFrameEvent contains information about a frame on a bus. The IFrameEvent is a specialization of IFrameEvent. Depending on the BusEventType the frame event was generated for a FlexRay, CAN/CAN FD or Ethernet (UDP) bus. The Raw property returns the raw payload data of the frame.

The Frame property references the frame as defined in the database. It may happen that the property returns null, if the frame was not specified in the database.

To get relevant bus information like CAN-Id, FlexRay or Ethernet Channel the IFrameEvent must be casted to appropriate ICanFrameEvent, IFlexRayFrameEvent or IEthernetFrameEvent.

Frame Interpreter

The Interpreter property IFrameInterpreter allows us to convert the raw bytes into pdu and signal information.

Example:

```
foreach (var pduInterpreter in frameEvent.Interpreter.GetPdus())
{
    foreach (var signal in pduInterpreter.GetSignals())
    {
        var signalValue = signal.GetValue();
        var interpretedValue = string.IsNullOrEmpty(signalValue.Unit)
            ? signalValue.InterpretedValue.ToString(CultureInfo.CurrentCulture)
            : signalValue.InterpretedValue.ToString(CultureInfo.CurrentCulture) +
            signalValue.Unit;
        var value = signalValue.IsTextDefined
            ? FormattableString.Invariant(
                $"{interpretedValue} [{signalValue.TextValue}]")
            : interpretedValue;
        System.Console.WriteLine($"{signal.Signal.ShortName}: {value}");
    }
}
```

Service Interpreter

SOME/IP service items (event, method, field) can be interpreted with the service interpreters. In this case the received IBusEvent will be a IEthernetServiceEvent, whose Interpreters property can be used to get the data types of the service items. The following service interpreters, which are derived from IDataTypeInterpreter can be used depending on the data types of the service item.

- ICommonDataTypeInterpreter
- IComplexDataTypeInterpreter
- IArrayDataInterpreter

Example:

We assume to have the following SOME/IP event with the name 'MyEvent' and want to get the physical value of the property 'C' from it:

```
MyEvent (Event)
|——A (Complex data type)
    |——B (Common data type)
    |——C (Common data type)
```

```
if (busEvent is IEthernetServiceEvent serviceEvent && serviceEvent.ServiceElementName == "MyEvent")
{
    if (serviceEvent.Interpreters.FirstOrDefault() is IComplexDataTypeInterpreter complexInterpreter)
    {
        var interpreter = complexInterpreter.Interpreters.FirstOrDefault(m => m.ShortName == "C");

     if (interpreter is ICommonDataTypeInterpreter commonDataTypeInterpreter)
        {
         Console.WriteLine(commonDataTypeInterpreter.GetValue().InterpretedValue);
        }
    }
}
```

Namespace FlexConfig.Sdk.Caching

Classes

Extension

Extension class to add database file caching.

Interfaces

IFileHandler

Interface for reading/writing optimized version of the database model.

Class Extension

Extension class to add database file caching.

Inheritance

Object
Extension

Inherited Members

Object.Equals(Object)
Object.Equals(Object, Object)
Object.GetHashCode()
Object.GetType()
Object.MemberwiseClone()
Object.ReferenceEquals(Object, Object)
Object.ToString()

Namespace: FlexConfig.Sdk.Caching

Assembly: FlexConfig.Sdk.Caching.dll

Syntax

```
public static class Extension
```

Methods

AddCaching(IDatabaseLoader, String)

Adds database file caching.

Declaration

```
public static IDatabaseLoader AddCaching(this IDatabaseLoader databaseLoader, string cacheDirectory)
```

Parameters

TYPE	NAME	DESCRIPTION
IDatabaseLoader	databaseLoader	The database loader instance.
String	cacheDirectory	The directory which should be used to load/save the cached database files.

Returns

TYPE	DESCRIPTION
IDatabaseLoader	The database loader.

Examples

```
FlexConfig.Sdk.DatabaseLoader.Instance.AddCaching(Path.Combine(System.Environment.CurrentDirectory,
"Caching"));
```

Interface IFileHandler

Interface for reading/writing optimized version of the database model.

Namespace: FlexConfig.Sdk.Caching

Assembly: FlexConfig.Sdk.Caching.dll

Syntax

```
public interface IFileHandler
```

Methods

Deserialize(Stream)

Deserialize the database from the given stream.

Declaration

```
IDatabase Deserialize(Stream stream)
```

Parameters

TYPE	NAME	DESCRIPTION
Stream	stream	The stream which contains the database.

Returns

TYPE	DESCRIPTION
IDatabase	The database model.

Serialize(IDatabase, Stream)

Serialize the database model to the stream.

Declaration

```
void Serialize(IDatabase database, Stream stream)
```

Parameters

TYPE	NAME	DESCRIPTION
IDatabase	database	The database model to serialize.
Stream	stream	The stream in which the database model should be serialized to.

Namespace FlexConfig.Sdk.Database

Classes

CanTriggering

A implementation of ICanTriggering interface.

DatabaseLoader

An implementation of the database loader.

FilterExtension

Extension class to add database filtering.

FlexRayTriggering

Implements the IFlexRayTriggering interface.

LanguageString

Supports multiple string based on the ui culture.

Interfaces

IAbsoluteTiming

Defines the timing information for a message on the flexray bus.

IArrayDimension

Defines the interface to get the array dimension information.

IBus

Defines the interface of a bus.

ICanTriggering

Defines the interface for a can triggering.

ICommonDataType

Defines the interface to get the information of a common data type. This interface implements the IDataType and ISignalSpecification.

IComplexDataType

Defines the interface to get the information of a complex data type. This interface implements the IDataType.

IComputationMethod

Basic interface for a computation method.

IContainedPduSpecification

Defines a IPduSpecification for a pdu contained inside a pdu container.

IContainerPduSpecification

Defines IPduSpecification for a container pdu.

IDatabase

Defines the interface for accessing the root information of automotive database.

IDatabaseEntity

Defines the base information of a database entity.

IDatabaseLoader

Handles the loading of automotive databases.

IDataType

Defines the interface to get the information of a data type. This interface implements the IDatabaseEntity.

IDataTypeDeclaration

Defines the interface to get the information of a data type declaration. This interface implements the IDatabaseEntity.

IElementLength

Defines the basic properties to define the length of an element.

IElementPosition

Defines the basic properties for the layout details of an element.

IEthernetConfiguration

Defines the properties of an ethernet configuration.

IEthernetTriggering

Defines the triggering for a ethernet message.

IEvent

Defines the interface for accessing SOME/IP event information. This interface implements the IMethod.

IEventGroup

Defines the interface for accessing SOME/IP event group information.

IField

Defines the interface for accessing SOME/IP field information.

IFilterExtension

Allows to filter elements so that this elements are not added to the database.

IFlexRayTriggering

Defines the triggering for a FlexRay message.

IFrame

Defines the interface for accessing frame information.

IFrameSpecification

Specifies the content for a IFrame.

IInterval

Defines an interval.

ILengthInfo

Interface ILengthInfo.

ILoaderExtension

Extension interface to change loading/saving databases.

IMethod

Defines the interface for accessing SOME/IP method information. This interface implements the IServiceElement.

IMultiplexedPduSpecification

Defines the interface for a multiplexed pdu specification.

IMultiplexerPduSpecification

Specifies a pdu as a multiplexer pdu.

IPdu

Defines the interface for describing a PDU.

IPduSpecification

Specifies the generic interface for a pdu.

IProvidedServiceInstance

Defines the interface to get the information of a provided service instance. This interface implements the IServiceInstance.

IScaleConstraint

Defines a interval with a validity flag.

IScalingInfo

Defines a scaling information.

ISecureCommunicationProperties

Contains the properties for secure communication.

ISegmentPosition

Describes an interface to specify segment positions.

IServiceElement

Defines the simple properties of a SOME/IP service element which can be an event, a method or a field.

IServiceInstance

Defines the properties of an service instance.

IServiceInterface

Defines the properties of a SOME/IP service interface.

ISignal

Defines the interface for a signal.

ISignalSpecification

Specifies the signal content.

IStandardPduSpecification

Specifies a standard pdu.

ITransportProtocol

Interface ITransportProtocol.

ITriggering

Defines the interface which triggers a message on the bus.

Enums

BusType

Defines the supported bus types.

ByteOrder

Defines the supported byte orderings.

ComputationType

Defines the possible computation methods.

ContainedCollectionSemanticType

Defines the possible collection semantics used for contained pdus.

ContainedTriggerType

Defines the supported trigger types for contained pdus.

ContainerAcceptMode

Defines the possible pdu container accept modes.

ContainerHeaderType

Defines the possible pdu container header types.

FlexRayChannel

The FlexRay channels.

IntervalType

Defines the supported type of intervals.

IpVersion

Defines the ip address version.

LengthInfoType

Defines the types of the length information (meta data).

ScaleConstraintValidity

Defines different validity types for an IInterval.

ServiceElementType

Defines the possible service element types.

SignalDataType

Defines the data type.

TransportProtocol

Enum TransportProtocol.

Enum BusType

Defines the supported bus types.

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public enum BusType
```

Fields

NAME	DESCRIPTION
BroadReach	BroadReach bus
CAN	CAN bus
CANFD	CAN-FD bus
Ethernet	Ethernet bus
FlexRay	FlexRay bus
Unknown	Unknown bus type.

Enum ByteOrder

Defines the supported byte orderings.

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public enum ByteOrder
```

Fields

NAME	DESCRIPTION
BigEndian	Big Endian byte order (Motorala).
Inherit	Use the byte order defined by the parent container.
LittleEndian	Little Endian byte order (Intel).

Class CanTriggering

A implementation of ICanTriggering interface.

Inheritance

Object
CanTriggering

Implements

ICanTriggering
ITriggering
IDatabaseEntity

Inherited Members

Object.Equals(Object)
Object.Equals(Object, Object)
Object.GetHashCode()
Object.GetType()
Object.MemberwiseClone()
Object.ReferenceEquals(Object, Object)
Object.ToString()

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
[DataContract]
public class CanTriggering : ICanTriggering, ITriggering, IDatabaseEntity
```

Constructors

CanTriggering()

Initializes a new instance of the CanTriggering class.

Declaration

```
public CanTriggering()
```

CanTriggering(Int32, Boolean)

Initializes a new instance of the CanTriggering class.

Declaration

```
public CanTriggering(int slotId, bool isExtendedId)
```

Parameters

TYPE	NAME	DESCRIPTION
Int32	slotId	The slot identifier.
Boolean	isExtendedId	if set to `true` [is extended identifier].

CanTriggering(Int32, Boolean, Boolean, Boolean)

Initializes a new instance of the CanTriggering class.

Declaration

```
public CanTriggering(int slotId, bool isExtendedId, bool canFdFrame, bool canFdBitRateSwitching)
```

Parameters

TYPE	NAME	DESCRIPTION
Int32	slotId	The slot identifier.
Boolean	isExtendedId	if set to `true` [is extended identifier].
Boolean	canFdFrame	if set to `true` this can triggering is a CAN FD triggering.
Boolean	canFdBitRateSwitching	if set to `true` the CAN FD bitrate switching is used.

Properties

CanFdBitRateSwitching

Gets a value indicating whether this can triggering supports the CAN FD bit rate switching.

Declaration

```
public bool CanFdBitRateSwitching { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

CanFdFrame

Gets a value indicating whether this can triggering supports the CAN FD Format.

Declaration

```
public bool CanFdFrame { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

CanIdentifier

Gets the CAN identifier.

Declaration

```
public int CanIdentifier { get; }
```

Property Value

TYPE	DESCRIPTION
Int32	

Description

Gets the description of the entity.

Declaration

```
public LanguageString Description { get; }
```

Property Value

TYPE	DESCRIPTION
LanguageString	

ExtendedId

Gets a value indicating whether this can triggering has an extended CAN identifier.

Declaration

```
public bool ExtendedId { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

Id

Gets the id of the entity.

Declaration

```
public string Id { get; }
```

Property Value

TYPE	DESCRIPTION
String	

LongName

Gets the long name of the entity.

Declaration

```
public LanguageString LongName { get; }
```

Property Value

TYPE	DESCRIPTION
LanguageString	

SenderEcus

Gets the sender ecus.

Declaration

```
public IEnumerable<string> SenderEcus { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<String>	The sender ecus.

ShortName

Gets the shortname of the entity.

Declaration

```
public string ShortName { get; }
```

Property Value

TYPE	DESCRIPTION
String	

Implements

ICanTriggering
ITriggering
IDatabaseEntity

Enum ComputationType

Defines the possible computation methods.

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public enum ComputationType
```

Fields

NAME	DESCRIPTION
Linear	Linear computation method.
TextTable	Text table computation method.
Undefined	Computation type not defined.

Enum ContainedCollectionSemanticType

Defines the possible collection semantics used for contained pdus.

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public enum ContainedCollectionSemanticType
```

Fields

NAME	DESCRIPTION
LastIsBest	The last is best semantic.
Queued	Queued semantic.

Enum ContainedTriggerType

Defines the supported trigger types for contained pdus.

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public enum ContainedTriggerType
```

Fields

NAME	DESCRIPTION
Always	Always.
Never	Never.

Enum ContainerAcceptMode

Defines the possible pdu container accept modes.

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public enum ContainerAcceptMode
```

Fields

NAME	DESCRIPTION
AcceptAll	Accept all pdus.
AcceptConfigured	Accept only the configured pdus.
NotDefined	Is not defined.

Enum ContainerHeaderType

Defines the possible pdu container header types.

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public enum ContainerHeaderType
```

Fields

NAME	DESCRIPTION
LongHeader	The long header.
NotDefined	Header type is not defined.
ShortHeader	The short header.

Class DatabaseLoader

An implementation of the database loader.

Inheritance

Object
DatabaseLoader

Implements

IDatabaseLoader

Inherited Members

Object.Equals(Object)
Object.Equals(Object, Object)
Object.GetHashCode()
Object.GetType()
Object.MemberwiseClone()
Object.ReferenceEquals(Object, Object)
Object.ToString()

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public class DatabaseLoader : IDatabaseLoader
```

Properties

FilterExtension

Gets or sets a filter extension to filter database elements (elements which should not be added to the database).

Declaration

```
public IFilterExtension FilterExtension { get; set; }
```

Property Value

TYPE	DESCRIPTION
IFilterExtension	

Instance

Gets the one and only instance of the database loader.

Declaration

```
public static IDatabaseLoader Instance { get; }
```

Property Value

TYPE	DESCRIPTION
IDatabaseLoader	

Methods

Load(Stream)

Load the database from a stream.

Declaration

```
[Licensing(Encryption = true, LicenseList = 2)]
public IDatabase Load(Stream stream)
```

Parameters

TYPE	NAME	DESCRIPTION
Stream	stream	The stream.

Returns

TYPE	DESCRIPTION
IDatabase	The loaded database.

Load(String)

Load the database.

Declaration

```
public IDatabase Load(string fileName)
```

Parameters

TYPE	NAME	DESCRIPTION
String	fileName	The file name.

Returns

TYPE	DESCRIPTION
IDatabase	The loaded database.

Implements

IDatabaseLoader

Extension Methods

Extension.AddCaching(IDatabaseLoader, String)
FilterExtension.AddFilter(IDatabaseLoader, IFilterExtension)

Class FilterExtension

Extension class to add database filtering.

Inheritance

Object
FilterExtension

Inherited Members

Object.Equals(Object)
Object.Equals(Object, Object)
Object.GetHashCode()
Object.GetType()
Object.MemberwiseClone()
Object.ReferenceEquals(Object, Object)
Object.ToString()

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public static class FilterExtension
```

Methods

AddFilter(IDatabaseLoader, IFilterExtension)

Adds database filtering support.

Declaration

```
public static IDatabaseLoader AddFilter(this IDatabaseLoader databaseLoader, IFilterExtension filterExtension)
```

Parameters

TYPE	NAME	DESCRIPTION
IDatabaseLoader	databaseLoader	The database loader instance.
IFilterExtension	filterExtension	The filter extension which should be used to filter the database files.

Returns

TYPE	DESCRIPTION
IDatabaseLoader	The database loader.

Examples

```
FlexConfig.Sdk.DatabaseLoader.Instance.AddFilter(new MyFilterExtension());
```

Enum FlexRayChannel

The FlexRay channels.

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public enum FlexRayChannel
```

Fields

NAME	DESCRIPTION
A	FlexRay Channel A
B	FlexRay Channel B
Both	FlexRay Channel A and B
None	None

Class FlexRayTriggering

Implements the IFlexRayTriggering interface.

Inheritance

Object
FlexRayTriggering

Implements

IFlexRayTriggering
ITriggering
IDatabaseEntity

Inherited Members

Object.Equals(Object)
Object.Equals(Object, Object)
Object.GetHashCode()
Object.GetType()
Object.MemberwiseClone()
Object.ReferenceEquals(Object, Object)
Object.ToString()

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
[DataContract]
public class FlexRayTriggering : IFlexRayTriggering, ITriggering, IDatabaseEntity
```

Constructors

FlexRayTriggering()

Initializes a new instance of the FlexRayTriggering class.

Declaration

```
public FlexRayTriggering()
```

FlexRayTriggering(Int16, FlexRayChannel)

Initializes a new instance of the FlexRayTriggering class.

Declaration

```
public FlexRayTriggering(short slotId, FlexRayChannel channel)
```

Parameters

TYPE	NAME	DESCRIPTION
Int16	slotId	The slot id.
FlexRayChannel	channel	The channel.

Properties

AbsoluteTimings

Gets the absolute timings.

Declaration

```
public IEnumerable<IAbsoluteTiming> AbsoluteTimings { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<IAbsoluteTiming>	The absolute timings.

Channel

Gets the channel.

Declaration

```
public FlexRayChannel Channel { get; }
```

Property Value

TYPE	DESCRIPTION
FlexRayChannel	The channel.

Description

Gets the description.

Declaration

```
public LanguageString Description { get; }
```

Property Value

TYPE	DESCRIPTION
LanguageString	The description.

Id

Gets the identifier.

Declaration

```
public string Id { get; }
```

Property Value

TYPE	DESCRIPTION
String	The identifier.

LongName

Gets the long name.

Declaration

```
public LanguageString LongName { get; }
```

Property Value

TYPE	DESCRIPTION
LanguageString	The long name.

SenderEcus

Gets the sender ecus.

Declaration

```
public IEnumerable<string> SenderEcus { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<String>	The sender ecus.

ShortName

Gets the short name.

Declaration

```
public string ShortName { get; }
```

Property Value

TYPE	DESCRIPTION
String	The short name.

Implements

IFlexRayTriggering
ITriggering
IDatabaseEntity

Interface IAbsoluteTiming

Defines the timing information for a message on the flexray bus.

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IAbsoluteTiming
```

Properties

BaseCycle

Gets the base cycle.

Declaration

```
short BaseCycle { get; }
```

Property Value

TYPE	DESCRIPTION
Int16	The base cycle.

CycleRepetition

Gets the cycle repetition.

Declaration

```
short CycleRepetition { get; }
```

Property Value

TYPE	DESCRIPTION
Int16	The cycle repetition.

SlotId

Gets the slot identifier.

Declaration

```
short SlotId { get; }
```

Property Value

TYPE	DESCRIPTION
Int16	The slot identifier.

Interface IArrayDimension

Defines the interface to get the array dimension information.

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IArrayDimension
```

Properties

BitAlignment

Gets the alignment of the array dimension.

Declaration

```
uint BitAlignment { get; }
```

Property Value

TYPE	DESCRIPTION
UInt32	

Remarks

This applies to array dimensions with dynamic sizes.

If e.g. 32bit alignment is specified, zero-padding-bits have to be filled from the end of the dimension to the next bigger 32bit block.

Dimension

Gets the dimension of the array for which the lower and upper bounds are specified.

Declaration

```
uint Dimension { get; }
```

Property Value

TYPE	DESCRIPTION
UInt32	

Remarks

E.g. a[5][5] has two dimensions with the upper and lower limit set to 5.

IsDynamic

Gets a value indicating whether this instance is a dynamic array.

Declaration

```
bool IsDynamic { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	`true` if this instance is dynamic; otherwise, `false`.

MaxSize

Gets the maximum count of elements in this array dimension.

Declaration

```
uint MaxSize { get; }
```

Property Value

TYPE	DESCRIPTION
UInt32	

MinSize

Gets the minimum count of elements in this array dimension.

Declaration

```
uint MinSize { get; }
```

Property Value

TYPE	DESCRIPTION
UInt32	

Interface IBus

Defines the interface of a bus.

Inherited Members

IDatabaseEntity.Id
IDatabaseEntity.ShortName
IDatabaseEntity.LongName
IDatabaseEntity.Description

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IBus : IDatabaseEntity
```

Properties

BusParameters

Gets the bus parameters. The values which will be returned depends on the bus type. For FlexRay the parameters (e.g. gdStaticSlot, gNumberOfMinislots,...) is returned.

Declaration

```
IEnumerable<KeyValuePair<string, double>> BusParameters { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<KeyValuePair<String, Double>>	

BusType

Gets the bus type.

Declaration

```
BusType BusType { get; }
```

Property Value

TYPE	DESCRIPTION
BusType	

Frames

Gets frames which are defined on this bus.

Declaration

```
IEnumerable<IFrame> Frames { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<IFrame>	

Pdus

Gets the pdus which are defined on this bus.

Declaration

```
IEnumerable<IPdu> Pdus { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<IPdu>	

ProvidedServices

Gets the provided service instances.

Declaration

```
IEnumerable<IProvidedServiceInstance> ProvidedServices { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<IProvidedServiceInstance>	The provided services.

Speed

Gets the bus speed in Bits per seconds.

Declaration

```
double Speed { get; }
```

Property Value

TYPE	DESCRIPTION
Double	

Interface ICanTriggering

Defines the interface for a can triggering.

Inherited Members

IDatabaseEntity.Id
IDatabaseEntity.ShortName
IDatabaseEntity.LongName
IDatabaseEntity.Description

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface ICanTriggering : ITriggering, IDatabaseEntity
```

Properties

CanFdBitRateSwitching

Gets a value indicating whether this can triggering supports the CAN FD bit rate switching.

Declaration

```
bool CanFdBitRateSwitching { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

CanFdFrame

Gets a value indicating whether this can triggering supports the CAN FD Format.

Declaration

```
bool CanFdFrame { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

CanIdentifier

Gets the CAN identifier.

Declaration

```
int CanIdentifier { get; }
```

Property Value

TYPE	DESCRIPTION

TYPE	DESCRIPTION
Int32	

ExtendedId

Gets a value indicating whether this can triggering has an extended CAN identifier.

Declaration

```
bool ExtendedId { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

SenderEcus

Gets the sender ecus.

Declaration

```
IEnumerable<string> SenderEcus { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<String>	The sender ecus.

Interface ICommonDataType

Defines the interface to get the information of a common data type. This interface implements the IDataType and ISignalSpecification.

Inherited Members

ISignalSpecification.CodedDataType
ISignalSpecification.PhysicalDataType
ISignalSpecification.ComputationMethods
IDatabaseEntity.Id
IDatabaseEntity.ShortName
IDatabaseEntity.LongName
IDatabaseEntity.Description
IElementLength.BitLength

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface ICommonDataType : IDataType, ISignalSpecification, IDatabaseEntity, IElementLength
```

Interface IComplexDataType

Defines the interface to get the information of a complex data type. This interface implements the IDataType.

Inherited Members

IDatabaseEntity.Id
IDatabaseEntity.ShortName
IDatabaseEntity.LongName
IDatabaseEntity.Description

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IComplexDataType : IDataType, IDatabaseEntity
```

Properties

Members

Gets the ordered members (sub data types) of the data type.

Declaration

```
IEnumerable<IDataTypeDeclaration> Members { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<IDataTypeDeclaration>	The members.

Interface IComputationMethod

Basic interface for a computation method.

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IComputationMethod
```

Properties

CodedValueValid

Gets the range where the coded value is valid to be used for this computation method.

Declaration

```
IInterval CodedValueValid { get; }
```

Property Value

TYPE	DESCRIPTION
IInterval	

PhysicalConstraints

Gets the physical constraints.

Declaration

```
IEnumerable<IScaleConstraint> PhysicalConstraints { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<IScaleConstraint>	The physical constraints.

ScalingConstraints

Gets the scaling constraints.

Declaration

```
IEnumerable<IScaleConstraint> ScalingConstraints { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<IScaleConstraint>	The scaling constraints.

Scalings

Gets the scalings.

Declaration

```
IEnumerable<IScalingInfo> Scalings { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<IScalingInfo>	The scalings.

Type

Gets the computation type.

Declaration

```
ComputationType Type { get; }
```

Property Value

TYPE	DESCRIPTION
ComputationType	

Unit

Gets the unit.

Declaration

```
string Unit { get; }
```

Property Value

TYPE	DESCRIPTION
String	

Interface IContainedPduSpecification

Defines a IPduSpecification for a pdu contained inside a pdu container.

Inherited Members

IPduSpecification.SecureCommunicationProperties
IDatabaseEntity.Id
IDatabaseEntity.ShortName
IDatabaseEntity.LongName
IDatabaseEntity.Description
IElementLength.BitLength

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IContainedPduSpecification : IPduSpecification, IDatabaseEntity, IElementLength
```

Properties

CollectionType

Gets the collection type.

Declaration

```
ContainedCollectionSemanticType CollectionType { get; }
```

Property Value

TYPE	DESCRIPTION
ContainedCollectionSemanticType	

LongHeader

Gets the long header value if defined.

Declaration

```
uint? LongHeader { get; }
```

Property Value

TYPE	DESCRIPTION
Nullable<UInt32>	

Offset

Gets the offset.

Declaration

```
uint? Offset { get; }
```

Property Value

TYPE	DESCRIPTION
Nullable<UInt32>	The offset.

Pdu

Gets the pdu.

Declaration

```
IPduSpecification Pdu { get; }
```

Property Value

TYPE	DESCRIPTION
IPduSpecification	The contained pdu.

ShortHeader

Gets the short header value if defined.

Declaration

```
uint? ShortHeader { get; }
```

Property Value

TYPE	DESCRIPTION
Nullable<UInt32>	

Timeout

Gets the timeout.

Declaration

```
uint Timeout { get; }
```

Property Value

TYPE	DESCRIPTION
UInt32	

TriggerType

Gets the trigger type.

Declaration

```
ContainedTriggerType TriggerType { get; }
```

Property Value

TYPE	DESCRIPTION
ContainedTriggerType	

Interface IContainerPduSpecification

Defines IPduSpecification for a container pdu.

Inherited Members

IPduSpecification.SecureCommunicationProperties
IDatabaseEntity.Id
IDatabaseEntity.ShortName
IDatabaseEntity.LongName
IDatabaseEntity.Description
IElementLength.BitLength

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IContainerPduSpecification : IPduSpecification, IDatabaseEntity, IElementLength
```

Properties

AcceptMode

Gets the accept mode.

Declaration

```
ContainerAcceptMode AcceptMode { get; }
```

Property Value

TYPE	DESCRIPTION
ContainerAcceptMode	The accept mode.

ContainedPdus

Gets the pdus which are included in the container pdu.

Declaration

```
IEnumerable<IContainedPduSpecification> ContainedPdus { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<IContainedPduSpecification>	The contained pdus.

HeaderType

Gets the type of the header.

Declaration

```
ContainerHeaderType HeaderType { get; }
```

Property Value

TYPE	DESCRIPTION
ContainerHeaderType	The type of the header.

IsContainedHeaderHighLow

Gets a value indicating whether this instance is contained header high low.

Declaration

```
bool IsContainedHeaderHighLow { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	`true` if this instance is contained header high low; otherwise, `false`.

ThresholdSize

Gets the size of the threshold.

Declaration

```
uint ThresholdSize { get; }
```

Property Value

TYPE	DESCRIPTION
UInt32	The size of the threshold.

TimeOut

Gets the time out.

Declaration

```
uint TimeOut { get; }
```

Property Value

TYPE	DESCRIPTION
UInt32	The time out.

Interface IDatabase

Defines the interface for accessing the root information of automotive database.

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IDatabase
```

Properties

Buses

Gets all buses which are defined in the database.

Declaration

```
IEnumerable<IBus> Buses { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<IBus>	

Frames

Gets all frames which are defined in the database.

Declaration

```
IEnumerable<IFrameSpecification> Frames { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<IFrameSpecification>	

ServiceInterfaces

Gets the service interfaces.

Declaration

```
IEnumerable<IServiceInterface> ServiceInterfaces { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<IServiceInterface>	The service instances.

Interface IDatabaseEntity

Defines the base information of a database entity.

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IDatabaseEntity
```

Properties

Description

Gets the description of the entity.

Declaration

```
LanguageString Description { get; }
```

Property Value

TYPE	DESCRIPTION
LanguageString	

Id

Gets the id of the entity.

Declaration

```
string Id { get; }
```

Property Value

TYPE	DESCRIPTION
String	

LongName

Gets the long name of the entity.

Declaration

```
LanguageString LongName { get; }
```

Property Value

TYPE	DESCRIPTION
LanguageString	

ShortName

Gets the shortname of the entity.

Declaration

```
string ShortName { get; }
```

Property Value

TYPE	DESCRIPTION
String	

Interface IDatabaseLoader

Handles the loading of automotive databases.

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IDatabaseLoader
```

Methods

Load(Stream)

Load the database from a stream.

Declaration

```
IDatabase Load(Stream stream)
```

Parameters

TYPE	NAME	DESCRIPTION
Stream	stream	The stream.

Returns

TYPE	DESCRIPTION
IDatabase	The loaded database.

Load(String)

Load the database.

Declaration

```
IDatabase Load(string fileName)
```

Parameters

TYPE	NAME	DESCRIPTION
String	fileName	The file name.

Returns

TYPE	DESCRIPTION
IDatabase	The loaded database.

Extension Methods

Extension.AddCaching(IDatabaseLoader, String)
FilterExtension.AddFilter(IDatabaseLoader, IFilterExtension)

Interface IDataType

Defines the interface to get the information of a data type. This interface implements the IDatabaseEntity.

Inherited Members

IDatabaseEntity.Id
IDatabaseEntity.ShortName
IDatabaseEntity.LongName
IDatabaseEntity.Description

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IDataType : IDatabaseEntity
```

Interface IDataTypeDeclaration

Defines the interface to get the information of a data type declaration. This interface implements the IDatabaseEntity.

Inherited Members

IDatabaseEntity.Id
IDatabaseEntity.ShortName
IDatabaseEntity.LongName
IDatabaseEntity.Description

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IDataTypeDeclaration : IDatabaseEntity
```

Properties

ArrayDimensions

Gets the dimensions of an possibly multi-dimensional array.

Declaration

```
IEnumerable<IArrayDimension> ArrayDimensions { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<IArrayDimension>	

BitLength

Gets the bit length for a data type declaration without considering the arrays.

Declaration

```
uint BitLength { get; }
```

Property Value

TYPE	DESCRIPTION
UInt32	The bit length.

BitPosition

Gets the bit position.

Declaration

```
uint BitPosition { get; }
```

Property Value

TYPE	DESCRIPTION
UInt32	The bit position.

DataId

Gets the data identifier of the data members.

Declaration

```
int DataId { get; }
```

Property Value

TYPE	DESCRIPTION
Int32	The data identifier. '-1' means the member has no identifier.

DataType

Gets the data type of this declaration.

Declaration

```
IDataType DataType { get; }
```

Property Value

TYPE	DESCRIPTION
IDataType	

Index

Gets the index to bring the members in a desirable logical order.

Declaration

```
int Index { get; }
```

Property Value

TYPE	DESCRIPTION
Int32	

Remarks

The Position defines the order of the elements in the memory (compared to the bit position).

The INDEX is used for unions.

IsArray

Gets a value indicating whether this data type declaration defines an array.

Declaration

```
bool IsArray { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	`true` if this data type declaration defines an array; otherwise, `false`.

IsDynamic

Gets a value indicating whether this instance is dynamic.

Declaration

```
bool IsDynamic { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	`true` if this instance is dynamic; otherwise, `false`.

Mandatory

Gets a value indicating whether the complex data type member is mandatory.

Declaration

```
bool Mandatory { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

Position

Gets the position of the complex data type member within the complex data type.

Declaration

```
uint Position { get; }
```

Property Value

TYPE	DESCRIPTION
UInt32	

Remarks

The Position defines the order of the elements in the memory (compared to the bit position).

The INDEX is used for unions.

TotalBitLength

Gets the total bit length for a data type declaration. Normally TotalBitLength and BitLength properties are equal. TotalBitLength are considered by arrays.

Declaration

```
uint TotalBitLength { get; }
```

Property Value

TYPE	DESCRIPTION
UInt32	The bit length.

Interface IElementLength

Defines the basic properties to define the length of an element.

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IElementLength
```

Properties

BitLength

Gets the bit length of this element.

Declaration

```
uint BitLength { get; }
```

Property Value

TYPE	DESCRIPTION
UInt32	

Interface IElementPosition

Defines the basic properties for the layout details of an element.

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IElementPosition
```

Properties

BitPosition

Gets the bit position of the element inside the container. A bit position of -1 indicates an invalid position.

Declaration

```
int BitPosition { get; }
```

Property Value

TYPE	DESCRIPTION
Int32	

ByteOrder

Gets they byte order used for this element.

Declaration

```
ByteOrder ByteOrder { get; }
```

Property Value

TYPE	DESCRIPTION
ByteOrder	

Interface IEthernetConfiguration

Defines the properties of an ethernet configuration.

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IEthernetConfiguration
```

Properties

Ecu

Gets the ecu of the service instance.

Declaration

```
string Ecu { get; }
```

Property Value

TYPE	DESCRIPTION
String	The ecu.

IpAddress

Gets the IP address.

Declaration

```
IPAddress IpAddress { get; }
```

Property Value

TYPE	DESCRIPTION
IPAddress	

IpVersion

Gets the used IP version.

Declaration

```
IpVersion IpVersion { get; }
```

Property Value

TYPE	DESCRIPTION
IpVersion	

TransportProtocol

Gets the transport protocol.

Declaration

```
ITransportProtocol TransportProtocol { get; }
```

Property Value

TYPE	DESCRIPTION
ITransportProtocol	The transport protocol.

Vlan

Gets the Vlan tag of the service instance.

Declaration

```
uint Vlan { get; }
```

Property Value

TYPE	DESCRIPTION
UInt32	The vlan.

Interface IEthernetTriggering

Defines the triggering for a ethernet message.

Inherited Members

IDatabaseEntity.Id
IDatabaseEntity.ShortName
IDatabaseEntity.LongName
IDatabaseEntity.Description

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IEthernetTriggering : ITriggering, IDatabaseEntity
```

Properties

HeaderId

Gets the header identifier.

Declaration

```
uint? HeaderId { get; }
```

Property Value

TYPE	DESCRIPTION
Nullable<UInt32>	The header identifier.

ReceiverEthernetSettings

Gets the receiver ethernet settings.

Declaration

```
IEnumerable<IEthernetConfiguration> ReceiverEthernetSettings { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<IEthernetConfiguration>	The receiver ethernet settings.

SenderEthernetSettings

Gets the sender ethernet settings.

Declaration

```
IEnumerable<IEthernetConfiguration> SenderEthernetSettings { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<IEthernetConfiguration>	The sender ethernet settings.

Vlan

Gets the vlan.

Declaration

```
uint Vlan { get; }
```

Property Value

TYPE	DESCRIPTION
UInt32	The vlan.

Interface IEvent

Defines the interface for accessing SOME/IP event information. This interface implements the IMethod.

Inherited Members

IMethod.InputParameters
IMethod.OutputParameters
IMethod.LengthInfos
IMethod.ByteOrder
IMethod.TransformerLengthInput
IMethod.TransformerLengthOutput
IServiceElement.MethodId
IServiceElement.ServiceElementType
IDatabaseEntity.Id
IDatabaseEntity.ShortName
IDatabaseEntity.LongName
IDatabaseEntity.Description

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IEvent : IMethod, IServiceElement, IDatabaseEntity
```

Properties

IsHeaderReduced

Gets a value indicating whether this instance has a reduced SOME/IP header.

Declaration

```
bool IsHeaderReduced { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	`true` if this instance has a reduce header; otherwise, `false`.

Interface IEventGroup

Defines the interface for accessing SOME/IP event group information.

Inherited Members

IDatabaseEntity.Id
IDatabaseEntity.ShortName
IDatabaseEntity.LongName
IDatabaseEntity.Description

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IEventGroup : IDatabaseEntity
```

Properties

EventIds

Gets the event ids.

Declaration

```
IEnumerable<uint> EventIds { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<UInt32>	The event ids.

NotifierIds

Gets the notifier ids.

Declaration

```
IEnumerable<uint> NotifierIds { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<UInt32>	The notifier ids.

Interface IField

Defines the interface for accessing SOME/IP field information.

Inherited Members

IDataTypeDeclaration.DataType
IDataTypeDeclaration.Position
IDataTypeDeclaration.Index
IDataTypeDeclaration.Mandatory
IDataTypeDeclaration.IsArray
IDataTypeDeclaration.IsDynamic
IDataTypeDeclaration.ArrayDimensions
IDataTypeDeclaration.BitPosition
IDataTypeDeclaration.TotalBitLength
IDataTypeDeclaration.BitLength
IDataTypeDeclaration.DataId
IDatabaseEntity.Id
IDatabaseEntity.ShortName
IDatabaseEntity.LongName
IDatabaseEntity.Description

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IField : IDataTypeDeclaration, IDatabaseEntity
```

Properties

ByteOrder

Gets the byte order for the data types.

Declaration

```
ByteOrder ByteOrder { get; }
```

Property Value

TYPE	DESCRIPTION
ByteOrder	The byte order.

GetterMethodId

Gets the getter.

Declaration

```
int GetterMethodId { get; }
```

Property Value

TYPE	DESCRIPTION

TYPE	DESCRIPTION
Int32	The getter.

LengthInfos

Gets the length infos for the dynamic items.

Declaration

```
IEnumerable<ILengthInfo> LengthInfos { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<ILengthInfo>	The length infos.

NotifierMethodId

Gets the notifier.

Declaration

```
int NotifierMethodId { get; }
```

Property Value

TYPE	DESCRIPTION
Int32	The notifier.

SetterMethodId

Gets the setter.

Declaration

```
int SetterMethodId { get; }
```

Property Value

TYPE	DESCRIPTION
Int32	The setter.

TransformerLengthGetter

Gets the transformer length getter.

Declaration

```
int TransformerLengthGetter { get; }
```

Property Value

TYPE	DESCRIPTION
Int32	The transformer length getter.

TransformerLengthNotifier

Gets the transformer length notifier.

Declaration

```
int TransformerLengthNotifier { get; }
```

Property Value

TYPE	DESCRIPTION
Int32	The transformer length notifier.

TransformerLengthSetter

Gets the transformer length setter.

Declaration

```
int TransformerLengthSetter { get; }
```

Property Value

TYPE	DESCRIPTION
Int32	The transformer length setter.

Interface IFilterExtension

Allows to filter elements so that this elements are not added to the database.

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IFilterExtension
```

Methods

ShouldBeAdded<T>(T)

Check if the given element should be added to the database).

Declaration

```
bool ShouldBeAdded<T>(T t)
    where T : class
```

Parameters

TYPE	NAME	DESCRIPTION
T	t	The instance to check.

Returns

TYPE	DESCRIPTION
Boolean	true if the instance should be added. Otherwise false.

Type Parameters

NAME	DESCRIPTION
T	The typ.

Interface IFlexRayTriggering

Defines the triggering for a FlexRay message.

Inherited Members

IDatabaseEntity.Id
IDatabaseEntity.ShortName
IDatabaseEntity.LongName
IDatabaseEntity.Description

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IFlexRayTriggering : ITriggering, IDatabaseEntity
```

Properties

AbsoluteTimings

Gets the absolute timings.

Declaration

```
IEnumerable<IAbsoluteTiming> AbsoluteTimings { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<IAbsoluteTiming>	The absolute timings.

Channel

Gets the channel.

Declaration

```
FlexRayChannel Channel { get; }
```

Property Value

TYPE	DESCRIPTION
FlexRayChannel	

SenderEcus

Gets the sender ecus.

Declaration

```
IEnumerable<string> SenderEcus { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<String>	The sender ecus.

Interface IFrame

Defines the interface for accessing frame information.

Inherited Members

IDatabaseEntity.Id
IDatabaseEntity.ShortName
IDatabaseEntity.LongName
IDatabaseEntity.Description

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IFrame : IDatabaseEntity
```

Properties

Specification

Gets the specification of the content of this frame.

Declaration

```
IFrameSpecification Specification { get; }
```

Property Value

TYPE	DESCRIPTION
IFrameSpecification	

Triggerings

Gets the information on which timing this element is transmitted or received.

Declaration

```
IEnumerable<ITriggering> Triggerings { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<ITriggering>	

Methods

GetTransmitableFrame()

Gets a frame which payload and signal values can be modified and transmitted with the Transmit(IDeviceBusConnector, ITransmitableFrame) method.

Declaration

```
ITransmitableFrame GetTransmitableFrame()
```

Returns

TYPE	DESCRIPTION
ITransmitableFrame	A frame which can be transmitted via Transmit(IDeviceBusConnector, ITransmitableFrame) method.

GetTransmitableFrame(ITriggering)

Gets a frame which payload and signal values can be modified and transmitted with the Transmit(IDeviceBusConnector, ITransmitableFrame) method.

Declaration

```
ITransmitableFrame GetTransmitableFrame(ITriggering triggering)
```

Parameters

TYPE	NAME	DESCRIPTION
ITriggering	triggering	The triggering used for transmission.

Returns

TYPE	DESCRIPTION
ITransmitableFrame	A frame which can be transmitted via Transmit(IDeviceBusConnector, ITransmitableFrame) method.

Interface IFrameSpecification

Specifies the content for a IFrame.

Inherited Members

IDatabaseEntity.Id
IDatabaseEntity.ShortName
IDatabaseEntity.LongName
IDatabaseEntity.Description
IElementLength.BitLength

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IFrameSpecification : IDatabaseEntity, IElementLength
```

Properties

Pdus

Gets the content of the IFrame.

Declaration

```
IEnumerable<IPdu> Pdus { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<IPdu>	

Interface IInterval

Defines an interval.

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IInterval
```

Properties

LowerLimit

Gets the lower limit value.

Declaration

```
double LowerLimit { get; }
```

Property Value

TYPE	DESCRIPTION
Double	

LowerLimitInterval

Gets the lower limit interval type.

Declaration

```
IntervalType LowerLimitInterval { get; }
```

Property Value

TYPE	DESCRIPTION
IntervalType	

UpperLimit

Gets the upper limit value.

Declaration

```
double UpperLimit { get; }
```

Property Value

TYPE	DESCRIPTION
Double	

UpperLimitInterval

Gets the upper limit interval type.

Declaration

```
IntervalType UpperLimitInterval { get; }
```

Property Value

TYPE	DESCRIPTION
IntervalType	

Interface ILengthInfo

Interface ILengthInfo.

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface ILengthInfo
```

Properties

Length

Gets the length.

Declaration

```
uint Length { get; }
```

Property Value

TYPE	DESCRIPTION
UInt32	The length.

LengthInfoType

Gets the type of the length information.

Declaration

```
LengthInfoType LengthInfoType { get; }
```

Property Value

TYPE	DESCRIPTION
LengthInfoType	The type of the length information.

Interface ILoaderExtension

Extension interface to change loading/saving databases.

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface ILoaderExtension
```

Methods

GetCacheId(Stream)

Get a unique id for the given stream.

Declaration

```
string GetCacheId(Stream stream)
```

Parameters

TYPE	NAME	DESCRIPTION
Stream	stream	The stream containing the input model.

Returns

TYPE	DESCRIPTION
String	Unique id to identify the contents of the stream.

Load(String)

Loads a model.

Declaration

```
IDatabase Load(string cacheId)
```

Parameters

TYPE	NAME	DESCRIPTION
String	cacheId	The unique id to identify the model to load.

Returns

TYPE	DESCRIPTION
IDatabase	The loaded database model.

Save(String, IDatabase)

Adds a database to the cache.

Declaration

```
void Save(string cacheId, IDatabase database)
```

Parameters

TYPE	NAME	DESCRIPTION
String	cacheId	The id to identify the database.
IDatabase	database	The database which should be saved.

Interface IMethod

Defines the interface for accessing SOME/IP method information. This interface implements the IServiceElement.

Inherited Members

IServiceElement.MethodId
IServiceElement.ServiceElementType
IDatabaseEntity.Id
IDatabaseEntity.ShortName
IDatabaseEntity.LongName
IDatabaseEntity.Description

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IMethod : IServiceElement, IDatabaseEntity
```

Properties

ByteOrder

Gets the byte order for the data types.

Declaration

```
ByteOrder ByteOrder { get; }
```

Property Value

TYPE	DESCRIPTION
ByteOrder	The byte order.

InputParameters

Gets the input parameters.

Declaration

```
IEnumerable<IDataTypeDeclaration> InputParameters { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<IDataTypeDeclaration>	The input parameters.

LengthInfos

Gets the length infos for the dynamic items.

Declaration

```
IEnumerable<ILengthInfo> LengthInfos { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<ILengthInfo>	The length infos.

OutputParameters

Gets the output parameters.

Declaration

```
IEnumerable<IDataTypeDeclaration> OutputParameters { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<IDataTypeDeclaration>	The output parameters.

TransformerLengthInput

Gets the payload offset input.

Declaration

```
int TransformerLengthInput { get; }
```

Property Value

TYPE	DESCRIPTION
Int32	The payload offset input.

TransformerLengthOutput

Gets the payload offset output.

Declaration

```
int TransformerLengthOutput { get; }
```

Property Value

TYPE	DESCRIPTION
Int32	The payload offset output.

Interface IMultiplexedPduSpecification

Defines the interface for a multiplexed pdu specification.

Inherited Members

IPduSpecification.SecureCommunicationProperties
IDatabaseEntity.Id
IDatabaseEntity.ShortName
IDatabaseEntity.LongName
IDatabaseEntity.Description
IElementLength.BitLength

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IMultiplexedPduSpecification : IPduSpecification, IDatabaseEntity, IElementLength
```

Properties

Specification

Gets the pdu specification for this switch code.

Declaration

```
IPduSpecification Specification { get; }
```

Property Value

TYPE	DESCRIPTION
IPduSpecification	

SwitchCode

Gets the switch code which is used to identify the pdu.

Declaration

```
ulong SwitchCode { get; }
```

Property Value

TYPE	DESCRIPTION
UInt64	

Interface IMultiplexerPduSpecification

Specifies a pdu as a multiplexer pdu.

Inherited Members

IPduSpecification.SecureCommunicationProperties
IDatabaseEntity.Id
IDatabaseEntity.ShortName
IDatabaseEntity.LongName
IDatabaseEntity.Description
IElementLength.BitLength

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IMultiplexerPduSpecification : IPduSpecification, IDatabaseEntity, IElementLength
```

Properties

DynamicPart

Gets the dynamic part of the multiplexer pdu.

Declaration

```
IEnumerable<IMultiplexedPduSpecification> DynamicPart { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<IMultiplexedPduSpecification>	

DynamicSegmentPositions

Gets the dynamic segment positions.

Declaration

```
IEnumerable<ISegmentPosition> DynamicSegmentPositions { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<ISegmentPosition>	The dynamic segment positions.

StaticPart

Gets the static part of a multiplexer pdu. If no static pdu is defined, this value will be set to null.

Declaration

```
IStandardPduSpecification StaticPart { get; }
```

Property Value

TYPE	DESCRIPTION
IStandardPduSpecification	

StaticSegmentPositions

Gets the static segment positions.

Declaration

```
IEnumerable<ISegmentPosition> StaticSegmentPositions { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<ISegmentPosition>	The static segment positions.

SwitchPart

Gets the switch part of the multiplexer pdu.

Declaration

```
ISignal SwitchPart { get; }
```

Property Value

TYPE	DESCRIPTION
ISignal	

Enum IntervalType

Defines the supported type of intervals.

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public enum IntervalType
```

Fields

NAME	DESCRIPTION
Closed	The interval is limited by the value given. The value itself is included.
Open	The interval is limited by the value given. The value itself is not included.

Interface IPdu

Defines the interface for describing a PDU.

Inherited Members

IDatabaseEntity.Id
IDatabaseEntity.ShortName
IDatabaseEntity.LongName
IDatabaseEntity.Description
IElementPosition.BitPosition
IElementPosition.ByteOrder
IElementLength.BitLength

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IPdu : IDatabaseEntity, IElementPosition, IElementLength
```

Properties

Specification

Gets the specification for the PDU.

Declaration

```
IPduSpecification Specification { get; }
```

Property Value

TYPE	DESCRIPTION
IPduSpecification	

Triggerings

Gets the information on which timing this element is transmitted or received.

Declaration

```
IEnumerable<ITriggering> Triggerings { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<ITriggering>	

UpdateBitPosition

Gets the bit position for the Update-Bit. Maybe null if no Update-Bit is used.

Declaration

```
uint? UpdateBitPosition { get; }
```

Property Value

TYPE	DESCRIPTION
Nullable<UInt32>	

Interface IPduSpecification

Specifies the generic interface for a pdu.

Inherited Members

IDatabaseEntity.Id
IDatabaseEntity.ShortName
IDatabaseEntity.LongName
IDatabaseEntity.Description
IElementLength.BitLength

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IPduSpecification : IDatabaseEntity, IElementLength
```

Properties

SecureCommunicationProperties

Gets the secure communication properties if the pdu is a secured pdu. if not null will be returned.

Declaration

```
ISecureCommunicationProperties SecureCommunicationProperties { get; }
```

Property Value

TYPE	DESCRIPTION
ISecureCommunicationProperties	

Interface IProvidedServiceInstance

Defines the interface to get the information of a provided service instance. This interface implements the IServiceInstance.

Inherited Members

IServiceInstance.InstanceId
IServiceInstance.EthernetConfiguration

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IProvidedServiceInstance : IServiceInstance
```

Properties

ConsumedServiceInstances

Gets the consumed service instances.

Declaration

```
IEnumerable<IServiceInstance> ConsumedServiceInstances { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<IServiceInstance>	The consumed service instances.

ServiceInterface

Gets the service interface.

Declaration

```
IServiceInterface ServiceInterface { get; }
```

Property Value

TYPE	DESCRIPTION
IServiceInterface	The service interface.

Enum IpVersion

Defines the ip address version.

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public enum IpVersion
```

Fields

NAME	DESCRIPTION
IPv4	The IP V4 is used.
IPv6	The IP V6 is used.

Interface IScaleConstraint

Defines a interval with a validity flag.

Inherited Members

IInterval.LowerLimitInterval
IInterval.LowerLimit
IInterval.UpperLimitInterval
IInterval.UpperLimit

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IScaleConstraint : IInterval
```

Properties

Validity

Gets the validity.

Declaration

```
ScaleConstraintValidity Validity { get; }
```

Property Value

TYPE	DESCRIPTION
ScaleConstraintValidity	The validity.

Interface IScalingInfo

Defines a scaling information.

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IScalingInfo
```

Properties

Denominator0

Gets the denominator0.

Declaration

```
double Denominator0 { get; }
```

Property Value

TYPE	DESCRIPTION
Double	The denominator0.

Numerator0

Gets the numerator0.

Declaration

```
double Numerator0 { get; }
```

Property Value

TYPE	DESCRIPTION
Double	The numerator0.

Numerator1

Gets the numerator1.

Declaration

```
double Numerator1 { get; }
```

Property Value

TYPE	DESCRIPTION
Double	The numerator1.

Range

Gets the range where this scaling info is valid.

Declaration

```
IInterval Range { get; }
```

Property Value

TYPE	DESCRIPTION
IInterval	The range.

Text

Gets the text.

Declaration

```
string Text { get; }
```

Property Value

TYPE	DESCRIPTION
String	The text.

Interface ISecureCommunicationProperties

Contains the properties for secure communication.

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface ISecureCommunicationProperties
```

Properties

AuthAlgorithm

Gets the authentication algorithm.

Declaration

```
string AuthAlgorithm { get; }
```

Property Value

TYPE	DESCRIPTION
String	

AuthDataFreshnessLength

Gets the authentication data freshness length.

Declaration

```
uint AuthDataFreshnessLength { get; }
```

Property Value

TYPE	DESCRIPTION
UInt32	

AuthDataFreshnessStartPosition

Gets the authentication data freshness start position.

Declaration

```
uint AuthDataFreshnessStartPosition { get; }
```

Property Value

TYPE	DESCRIPTION
UInt32	

AuthenticationBuildAttempts

Gets the authentication build attempts.

Declaration

```
uint AuthenticationBuildAttempts { get; }
```

Property Value

TYPE	DESCRIPTION
UInt32	

AuthenticationRetries

Gets the authentication retries.

Declaration

```
uint AuthenticationRetries { get; }
```

Property Value

TYPE	DESCRIPTION
UInt32	

AuthInfoTxLength

Gets the authentication info tx length.

Declaration

```
uint AuthInfoTxLength { get; }
```

Property Value

TYPE	DESCRIPTION
UInt32	

DataId

Gets the data id.

Declaration

```
uint DataId { get; }
```

Property Value

TYPE	DESCRIPTION
UInt32	

FreshnessCounterSyncAttempts

Gets the freshness counter sync attempts.

Declaration

```
uint FreshnessCounterSyncAttempts { get; }
```

Property Value

TYPE	DESCRIPTION
UInt32	

FreshnessTimestampTimePeriodFactor

Gets the freshness timestamp period factor.

Declaration

```
uint FreshnessTimestampTimePeriodFactor { get; }
```

Property Value

TYPE	DESCRIPTION
UInt32	

FreshnessValueId

Gets the freshness value id.

Declaration

```
uint FreshnessValueId { get; }
```

Property Value

TYPE	DESCRIPTION
UInt32	

FreshnessValueLength

Gets the freshness value length.

Declaration

```
uint FreshnessValueLength { get; }
```

Property Value

TYPE	DESCRIPTION
UInt32	

FreshnessValueTxLength

Gets the freshness value tx length.

Declaration

```
uint FreshnessValueTxLength { get; }
```

Property Value

TYPE	DESCRIPTION
UInt32	

KeyId

Gets the key id.

Declaration

```
uint KeyId { get; }
```

Property Value

TYPE	DESCRIPTION
UInt32	

MessageLinkLength

Gets the message link length.

Declaration

```
uint MessageLinkLength { get; }
```

Property Value

TYPE	DESCRIPTION
UInt32	

MessageLinkPosition

Gets the message link position.

Declaration

```
uint MessageLinkPosition { get; }
```

Property Value

TYPE	DESCRIPTION
UInt32	

SecondaryFreshnessValueId

Gets the secondary freshness value id.

Declaration

```
uint SecondaryFreshnessValueId { get; }
```

Property Value

TYPE	DESCRIPTION
UInt32	

UseFreshnessTimeStamp

Gets a value indicating whether freshness timestamp is used.

Declaration

```
bool UseFreshnessTimeStamp { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

Interface ISegmentPosition

Describes an interface to specify segment positions.

Inherited Members

IElementPosition.BitPosition
IElementPosition.ByteOrder
IElementLength.BitLength

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface ISegmentPosition : IElementPosition, IElementLength
```

Interface IServiceElement

Defines the simple properties of a SOME/IP service element which can be an event, a method or a field.

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IServiceElement
```

Properties

MethodId

Gets the method identifier.

Declaration

```
ushort MethodId { get; }
```

Property Value

TYPE	DESCRIPTION
UInt16	The method identifier.

ServiceElementType

Gets the type of the service element.

Declaration

```
ServiceElementType ServiceElementType { get; }
```

Property Value

TYPE	DESCRIPTION
ServiceElementType	The type of the service element.

Interface IServiceInstance

Defines the properties of an service instance.

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IServiceInstance
```

Properties

EthernetConfiguration

Gets the ethernet configuration.

Declaration

```
IEthernetConfiguration EthernetConfiguration { get; }
```

Property Value

TYPE	DESCRIPTION
IEthernetConfiguration	The ethernet configuration.

InstanceId

Gets the instance identifier of provided/consumed service.

Declaration

```
ushort InstanceId { get; }
```

Property Value

TYPE	DESCRIPTION
UInt16	The instance identifier.

Interface IServiceInterface

Defines the properties of a SOME/IP service interface.

Inherited Members

IDatabaseEntity.Id
IDatabaseEntity.ShortName
IDatabaseEntity.LongName
IDatabaseEntity.Description

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IServiceInterface : IDatabaseEntity
```

Properties

EventGroups

Gets the collection of the IEventGroups of this IServiceInterface.

Declaration

```
IEnumerable<IEventGroup> EventGroups { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<IEventGroup>	

Remarks

An IEventGroup hosts one or more IEvent functions.

A Client can only consume a whole group.

Events

Gets the collection of IMethod described within the IServiceInterface.

Declaration

```
IEnumerable<IEvent> Events { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<IEvent>	

Remarks

An event is a method that can be subscribed on.

A callback at the subscribers side is required.

Fields

Gets the collection of the attributes (IField) of the IServiceInterface.

Declaration

```
IEnumerable<IField> Fields { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<IField>	

Methods

Gets the collection of IMethod described within the IServiceInterface.

Declaration

```
IEnumerable<IMethod> Methods { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<IMethod>	

ServiceIdentifier

Gets the service identifier.

Declaration

```
ushort ServiceIdentifier { get; }
```

Property Value

TYPE	DESCRIPTION
UInt16	The service identifier.

Interface ISignal

Defines the interface for a signal.

Inherited Members

IDatabaseEntity.Id
IDatabaseEntity.ShortName
IDatabaseEntity.LongName
IDatabaseEntity.Description
IElementPosition.BitPosition
IElementPosition.ByteOrder
IElementLength.BitLength

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface ISignal : IDatabaseEntity, IElementPosition, IElementLength
```

Properties

Specification

Gets the specification used for the signal.

Declaration

```
ISignalSpecification Specification { get; }
```

Property Value

TYPE	DESCRIPTION
ISignalSpecification	

Interface ISignalSpecification

Specifies the signal content.

Inherited Members

IDatabaseEntity.Id
IDatabaseEntity.ShortName
IDatabaseEntity.LongName
IDatabaseEntity.Description
IElementLength.BitLength

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface ISignalSpecification : IDatabaseEntity, IElementLength
```

Properties

CodedDataType

Gets the data type of the coded value.

Declaration

```
SignalDataType CodedDataType { get; }
```

Property Value

TYPE	DESCRIPTION
SignalDataType	

ComputationMethods

Gets the computation methods.

Declaration

```
IReadOnlyList<IComputationMethod> ComputationMethods { get; }
```

Property Value

TYPE	DESCRIPTION
IReadOnlyList<IComputationMethod>	The computation methods.

PhysicalDataType

Gets the data type for the physical value.

Declaration

```
SignalDataType PhysicalDataType { get; }
```

Property Value

TYPE	DESCRIPTION
SignalDataType	

Interface IStandardPduSpecification

Specifies a standard pdu.

Inherited Members

IPduSpecification.SecureCommunicationProperties
IDatabaseEntity.Id
IDatabaseEntity.ShortName
IDatabaseEntity.LongName
IDatabaseEntity.Description
IElementLength.BitLength

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IStandardPduSpecification : IPduSpecification, IDatabaseEntity, IElementLength
```

Properties

Signals

Gets the signals which are included in the pdu.

Declaration

```
IEnumerable<ISignal> Signals { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<ISignal>	

Interface ITransportProtocol

Interface ITransportProtocol.

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface ITransportProtocol
```

Properties

PortDynamicAssigned

Gets a value indicating whether the source port is dynamically assigned.

Declaration

```
bool PortDynamicAssigned { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	`true` if the source port is dynamically assigned; otherwise, `false`.

PortNumber

Gets the port number.

Declaration

```
uint PortNumber { get; }
```

Property Value

TYPE	DESCRIPTION
UInt32	

Remarks

IT Transport-Protocol Port type defining the port (source or destination) for e.g. UDP and TCP.

A port number '0' means:

that no port number is defined (port is dynamically assigned).

parsing the port failed).

Protocol

Gets the protocol.

Declaration

```
TransportProtocol Protocol { get; }
```

Property Value

TYPE	DESCRIPTION
TransportProtocol	The protocol.

Interface ITriggering

Defines the interface which triggers a message on the bus.

Inherited Members

IDatabaseEntity.Id
IDatabaseEntity.ShortName
IDatabaseEntity.LongName
IDatabaseEntity.Description

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface ITriggering : IDatabaseEntity
```

Class LanguageString

Supports multiple string based on the ui culture.

Inheritance

Object

LanguageString

Inherited Members

Object.Equals(Object)

Object.Equals(Object, Object)

Object.GetHashCode()

Object.GetType()

Object.MemberwiseClone()

Object.ReferenceEquals(Object, Object)

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
[DataContract]
public class LanguageString
```

Properties

Item[String]

Gets or set the string based on the given language. If language is empty then the CurrentUICulture language is used.

Declaration

```
[IgnoreDataMember]
public string this[string language] { get; set; }
```

Parameters

TYPE	NAME	DESCRIPTION
String	language	The language. If null or empty then the the CurrentUICulture language is used.

Property Value

TYPE	DESCRIPTION
String	The string for the given language.

Value

Gets the string based on the CurrentUICulture. If no string for the current culture is found, the english culture will be used. If no english string can be found, than the first culture is used.

Declaration

```
[IgnoreDataMember]
public string Value { get; }
```

Property Value

TYPE	DESCRIPTION
String	

Methods

ToString()

Declaration

```
public override string ToString()
```

Returns

TYPE	DESCRIPTION
String	

Overrides

Object.ToString()

Enum LengthInfoType

Defines the types of the length information (meta data).

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public enum LengthInfoType
```

Fields

NAME	DESCRIPTION
ArrayLengthInput	The input array length. This is used by events, fields and input parameters of methods.
ArrayLengthOutput	The output array length. This is used by output parameters of methods.
StructLengthInput	The input structure length. This is used by events, fields and input parameters of methods.
StructLengthOutput	The output structure length. This is used by output parameters of methods.
UnionLengthInput	The input union length. This is used by events, fields and input parameters of methods.
UnionLengthOutput	The output union length. This is used by output parameters of methods.

Enum ScaleConstraintValidity

Defines different validity types for an IInterval.

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public enum ScaleConstraintValidity
```

Fields

NAME	DESCRIPTION
Error	Error
NotAvailable	Not available.
NotDefined	Not defined.
NotValid	Not valid
Other	Other
Valid	Valid.

Enum ServiceElementType

Defines the possible service element types.

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public enum ServiceElementType
```

Fields

NAME	DESCRIPTION
Event	The event
Field	The field
FieldGetter	The field getter
FieldNotifier	The notifier
FieldSetter	The setter
Method	The method
None	The none

Enum SignalDataType

Defines the data type.

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public enum SignalDataType
```

Fields

NAME	DESCRIPTION
FloatingPoint	A floating point value. Based on the bit length, this is 32 Bit float or a 64 Bit float type.
SignedInteger	A signed integer value.
SignedLong	The signed long
Undefined	The data type is not defined.
UnsignedInteger	A unsigned integer value
UnsignedLong	The unsigned long

Enum TransportProtocol

Enum TransportProtocol.

Namespace: FlexConfig.Sdk.Database

Assembly: FlexConfig.Sdk.dll

Syntax

```
public enum TransportProtocol
```

Fields

NAME	DESCRIPTION
Generic	The generic protocol
Http	The HTTP protocol
Ieee1722	The ieee1722 protocol
Rtp	The RTP protocol
Tcp	The TCP protocol
Udp	The UDP protocol

Namespace FlexConfig.Sdk.Devices

Classes

BusEventEventArgs

The event argument class for OnReceive handler.

ContainedMultiplexerPdu

Initializes a new ContainedMultiplexerPdu object.

ContainedStandardPdu

Initializes a new ContainedStandardPdu object.

DeviceManager

Implementation of IDeviceManager. Handles searching for connected devices and their access state.

Interfaces

IBusEvent

Defines the basic interface for events which is generated by a device. The events are received via OnReceive method.

ICanErrorEvent

Defines the interface for keeping information about a CAN/CAN-FD error event.

ICanFrameEvent

Defines a message which was received or transmitted on a CAN or CAN FD bus.

ICommunicationErrorEvent

Defines the interface to identifying communication error events on the bus.

IDevice

Defines the interface for a device which can be used to monitor frames from buses.

IDeviceBusConnector

Gets information about a device bus connector, e.g. which bus type it supports, on which physical port the bus is available.

IDeviceInfo

Defines the information from a device. This includes the type of device and which buses are supported by the device.

IDeviceManager

Handles the enumeration, opening and closing devices.

IEthernetEvent

Defines the interface for keeping information about decoded ethernet package.

IEthernetFrameEvent

Defines the interface for an ethernet frame event which is not related with a service element. Implements the IFrameEvent Implements the IEthernetEvent.

IEthernetServiceEvent

Defines the interface for an ethernet service elements such as events, methods and fields. Implements the IEthernetEvent.

IFlexRayErrorEvent

Defines the interface for keeping information about a FlexRay error event.

IFlexRayFrameEvent

Defines a message which was received or transmitted on a FlexRay bus.

IFlexRayStatusEvent

Defines the interface for keeping information about a FlexRay status event.

IFlexRaySyncInformationEvent

Contains information about a FlexRay synchronization.

IFrameEvent

Defines the interface for a frame event.

IMultiplexerSegment

The segment of a multiplexer PDU.

IStatusEvent

Defines the interface to identifying status events from the bus.

ISwitchCodeSignal

The switch code of a multiplexed PDU.

ITransmitableContainerPdu

The container PDU.

ITransmitableFrame

The frame to be transmitted.

ITransmitableMultiplexerPdu

The multiplexer PDU.

ITransmitablePdu

The PDU to be transmitted with its frame.

ITransmitableSignal

The signal to be transmitted with its PDU.

ITransmitableStaticPdu

The PDU in the static part of the multiplexer PDU.

ITransmitableSwitchedPdu

The switched PDU in a dynamic segment.

Enums

BusCanErrorType

Defines the possible can errors.

BusCommunicationControllerState

Defines the possible states for the bus communication controller.

BusCommunicationErrorType

Defines the possible bus communication error types.

BusEventType

Defines the possible bus event types. Bus events are reported via OnReceive handler.

BusFlexRayErrorType

Defines the possible errors reported by a IFlexRayErrorEvent.

BusFlexRayWakeupStatus

Defines the different flexray bus wakeup states reported by IFlexRayStatusEvent.

BusStatusFlag

Defines the possible bus status flags.

CcType

Defines the supported communication controller types.

ContainedPduType

Defines the contained PDU type.

DeviceBusConnectorName

Defines the names of a device bus connector. This names are used to identify on which hardware connector the specific bus is available.

DeviceType

Defines the different device types.

EthernetMessageType

Enum EthernetMessageType.

EthernetMode

Enum EthernetMode.

FrameDirection

Defines the possible direction for a message.

MultiplexerSegmentType

Defines the possible segment types for a segment in a multiplexer PDU.

Enum BusCanErrorType

Defines the possible can errors.

Namespace: FlexConfig.Sdk.Devices

Assembly: FlexConfig.Sdk.dll

Syntax

```
public enum BusCanErrorType
```

Fields

NAME	DESCRIPTION
AcknowledgeError	The message the CAN communication controller transmitted was not acknowledged by another node.
Bit0Error	During the transmission of a message, the device wanted to send a dominant level (data or identifier bit logical value 0), but the monitored bus value was recessive (data or identifier bit logical value 1).
Bit1Error	During the transmission of a message (with the exception of the arbitration field), the device wanted to send a recessive level (bit of logical value 1), but the monitored bus value was dominant (bit of logical value 0).
CrcError	The CRC check sum was incorrect in the message received, the CRC received for an incoming message does not match with the calculated CRC for the received data.
FormError	A fixed format part of a received frame has the wrong format.
None	No error occurred.
ParityError	The Parity Check Mechanism has detected a parity error in the Message RAM.
StuffError	More than 5 equal bits in a sequence have occurred in a part of a received message where this is not allowed.

Enum BusCommunicationControllerState

Defines the possible states for the bus communication controller.

Namespace: FlexConfig.Sdk.Devices

Assembly: FlexConfig.Sdk.dll

Syntax

```
public enum BusCommunicationControllerState
```

Fields

NAME	DESCRIPTION
BusOff	Communication controller is in bus off state.
Config	Communication controller is in configuration state.
ErrorPassive	Communication controller is in Passive state. No CAN messages can be sent anymore.
ErrorWarning	Communication controller is in error warning state. At least one of the error counters has reached the error warning limit of 96.
NormalActive	Communication controller is in normal active state.
Unknown	Communication controller state is unknown.

Enum BusCommunicationErrorType

Defines the possible bus communication error types.

Namespace: FlexConfig.Sdk.Devices

Assembly: FlexConfig.Sdk.dll

Syntax

```
public enum BusCommunicationErrorType
```

Fields

NAME	DESCRIPTION
BusTransmissionError	Bus transmission error detected.
BusTransmissionImpossible	Bus transmission impossible detected.
TransmitBufferOverflow	Transmit buffer overflow detected.
Unknown	Unknown error detected.

Class BusEventEventArgs

The event argument class for OnReceive handler.

Inheritance

Object
EventArgs
BusEventEventArgs

Inherited Members

EventArgs.Empty
Object.Equals(Object)
Object.Equals(Object, Object)
Object.GetHashCode()
Object.GetType()
Object.MemberwiseClone()
Object.ReferenceEquals(Object, Object)
Object.ToString()

Namespace: FlexConfig.Sdk.Devices

Assembly: FlexConfig.Sdk.dll

Syntax

```
public class BusEventEventArgs : EventArgs
```

Constructors

BusEventEventArgs(IEnumerable<IBusEvent>)

Initializes a new instance of the BusEventEventArgs class.

Declaration

```
public BusEventEventArgs(IEnumerable<IBusEvent> busEvents)
```

Parameters

TYPE	NAME	DESCRIPTION
IEnumerable<IBusEvent>	busEvents	The bus events.

Properties

Events

Gets the bus events.

Declaration

```
public IEnumerable<IBusEvent> Events { get; }
```

Property Value

TYPE	DESCRIPTION

TYPE	DESCRIPTION
IEnumerable<IBusEvent>	The bus interpreters.

Enum BusEventType

Defines the possible bus event types. Bus events are reported via OnReceive handler.

Namespace: FlexConfig.Sdk.Devices

Assembly: FlexConfig.Sdk.dll

Syntax

```
public enum BusEventType
```

Fields

NAME	DESCRIPTION
Can	Event contains a CAN frame was received or transmitted. The generated event is ICanFrameEvent.
CanError	Event contains a CAN/CAN FD error event. The generated event is ICanErrorEvent.
CommunicationError	Event contain a communication error event. The generated event is ICommunicationErrorEvent.
EthernetFrame	Event contains an Ethernet frame with PDUs and signals, which are not related with service elements. The generated event is IEthernetFrameEvent.
EthernetService	Event contains an Ethernet SOME/IP service. The generated event is IEthernetServiceEvent.
FlexRay	Event contains a FlexRay frame was received or transmitted. The generated event is IFlexRayFrameEvent.
FlexRayError	Event contains a FlexRay error event. The generated event is IFlexRayErrorEvent.
FlexRayStatus	Events contains a FlexRay status event. The generated event is IFlexRayStatusEvent.
FlexRaySyncInformation	Event contains information about FlexRay synchronization. The generated event is IFlexRaySyncInformationEvent.

Enum BusFlexRayErrorType

Defines the possible errors reported by a IFlexRayErrorEvent.

Namespace: FlexConfig.Sdk.Devices

Assembly: FlexConfig.Sdk.dll

Syntax

```
public enum BusFlexRayErrorType
```

Fields

NAME	DESCRIPTION
ClockCorrectionFailure	Additional information are described in the structure ErrClockCorrectionFailureInfo
Content	A content error was observed in the configured slot (frame is semantically incorrect). (FlexRay Protocol Specification: 'vSS!ContentErrorA' or 'vSS!ContentErrorB' depends on Channel) Additional information described in the structure ErrSlotInfo
EmptyFifoAccess	Indicates ErrEmptyFIFOAccess
FlexcardOverflow	FlexCard buffer overflow. This error occurs if the application was too slow to receive and process the packets. If the FlexCard is configured to stop the monitoring it is necessary to stop and start the monitoring again. Else the FlexCard continue the monitoring when an amount of free RAM space is available again. In such a case the FlexCard loses packets.
IllegalInputBufferAccess	Indicates ErrIllegalInputBufferAccess
IllegalOutputbufferAccess	Indicates ErrIllegalOutputbufferAccess
LatestTransmitViolation	Additional information are described in the structure ErrSlotInfo
None	No error occurred
ParityError	Internal E-Ray error. No additional information is available
PocErrorModeChanged	Protocol Operation Control error. Additional information are described in the structure ErrPOCErrorModeChangedInfo
ReceiveFifoOverrun	The following four errors are internal FlexCard errors. For these errors no additional information exists (ErrorPacket.AdditionalInfo is not valid)

NAME	DESCRIPTION
SlotBoundaryViolation	A slot boundary violation was observed. (FlexRay Protocol Specification: 'vSS!BViolationA' or 'vSS!BViolationB' depends on Channel) Additional information described in the structure ErrSlotInfo
SlotBoundaryViolationNit	Slot boundary violation in network idle time was observed. Additional information are described in the structure ErrSlotInfo.
SlotBoundaryViolationSw	Slot boundary violation in symbol window was observed. Additional information are described in the structure ErrSlotInfo.
SyncFrameOverflow	Additional information are described in the structure ErrSyncFramesInfo
SyncFramesBelowMinimum	Additional information are described in the structure ErrSyncFramesInfo
Syntax	A syntax error was observed in the configured slot (frame is syntactically incorrect). (FlexRay Protocol Specification: 'vSS!SyntaxErrorA' or 'vSS!SyntaxErrorB' depends on Channel) Additional information described in the structure ErrSlotInfo
SyntaxNit	Syntax error in network idle time was observed. Additional information are described in the structure ErrSlotInfo.
SyntaxSw	Syntax error in symbol window was observed. Additional information are described in the structure ErrSlotInfo.
TransmissionAcrossBoundary	Additional information are described in the structure ErrSlotInfo
TransmissionConflictSw	Transmission conflict in symbol window was observed. Additional information are described in the structure ErrSlotInfo.

Enum BusFlexRayWakeupStatus

Defines the different flexray bus wakeup states reported by IFlexRayStatusEvent.

Namespace: FlexConfig.Sdk.Devices

Assembly: FlexConfig.Sdk.dll

Syntax

```
public enum BusFlexRayWakeupStatus
```

Fields

NAME	DESCRIPTION
CollisionHeader	FlexRay Protocol Specification: COLLISION_HEADER.
CollisionUnknown	FlexRay Protocol Specification: COLLISION_UNKNOWN.
ReceiveHeader	FlexRay Protocol Specification: RECEIVE_HEADER.
ReceiveWup	FlexRay Protocol Specification: RECEIVE_WUP.
Transmitted	FlexRay Protocol Specification: TRANSMITTED.
Undefined	FlexRay Protocol Specification: UNDEFINED.
WakeupStatusCollisionWup	FlexRay Protocol Specification: COLLISION_WUP.

Enum BusStatusFlag

Defines the possible bus status flags.

Namespace: FlexConfig.Sdk.Devices

Assembly: FlexConfig.Sdk.dll

Syntax

```
public enum BusStatusFlag
```

Fields

NAME	DESCRIPTION
StatusCollisionAvoidanceSymbol	Collision avoidance symbol was received
StatusMtsReceivedonChannelA	Media Access Test Symbol received on Channel A
StatusMtsReceivedonChannelB	Media Access Test Symbol received on Channel B
StatusNone	No status change.
StatusStartupCompletedSuccessfully	Start up has been successfully completed.
StatusWakeupPatternChannelA	Wakeup pattern received on Channel A
StatusWakeupPatternChannelB	Wakeup pattern received on Channel B
StatusWakeupStatus	Wakeup status has changed.

Enum CcType

Defines the supported communication controller types.

Namespace: FlexConfig.Sdk.Devices

Assembly: FlexConfig.Sdk.dll

Syntax

```
public enum CcType
```

Fields

NAME	DESCRIPTION
DCan	Communication controller is a Bosch D-CAN.
ERay	Communication controller is a Bosch E-Ray (FlexRay).
Ethernet	Cc type Ethernet.
MCan	Communication controller is a Bosch M-CAN.
Unknown	Unknown cc type.

Class ContainedMultiplexerPdu

Initializes a new ContainedMultiplexerPdu object.

Inheritance

Object

ContainedMultiplexerPdu

Inherited Members

Object.Equals(Object)
Object.Equals(Object, Object)
Object.GetHashCode()
Object.GetType()
Object.MemberwiseClone()
Object.ReferenceEquals(Object, Object)
Object.ToString()

Namespace: FlexConfig.Sdk.Devices

Assembly: FlexConfig.Sdk.dll

Syntax

```
public class ContainedMultiplexerPdu
```

Properties

PduType

Gets the PDU type.

Declaration

```
public ContainedPduType PduType { get; }
```

Property Value

TYPE	DESCRIPTION
ContainedPduType	

Specification

Gets the specification.

Declaration

```
public IMultiplexerPduSpecification Specification { get; }
```

Property Value

TYPE	DESCRIPTION
IMultiplexerPduSpecification	

Enum ContainedPduType

Defines the contained PDU type.

Namespace: FlexConfig.Sdk.Devices

Assembly: FlexConfig.Sdk.dll

Syntax

```
public enum ContainedPduType
```

Fields

NAME	DESCRIPTION
MultiplexerPdu	The contained PDU is a multiplexer PDU.
StandardPdu	The contained PDU is a standard PDU.

Class ContainedStandardPdu

Initializes a new ContainedStandardPdu object.

Inheritance

Object

ContainedStandardPdu

Inherited Members

Object.Equals(Object)
Object.Equals(Object, Object)
Object.GetHashCode()
Object.GetType()
Object.MemberwiseClone()
Object.ReferenceEquals(Object, Object)
Object.ToString()

Namespace: FlexConfig.Sdk.Devices

Assembly: FlexConfig.Sdk.dll

Syntax

```
public class ContainedStandardPdu
```

Properties

PduType

Gets the PDU type.

Declaration

```
public ContainedPduType PduType { get; }
```

Property Value

TYPE	DESCRIPTION
ContainedPduType	

Specification

Gets the specification.

Declaration

```
public IStandardPduSpecification Specification { get; }
```

Property Value

TYPE	DESCRIPTION
IStandardPduSpecification	

Enum DeviceBusConnectorName

Defines the names of a device bus connector. This names are used to identify on which hardware connector the specific bus is available.

Namespace: FlexConfig.Sdk.Devices

Assembly: FlexConfig.Sdk.dll

Syntax

```
public enum DeviceBusConnectorName
```

Remarks

For the FlexDevice family the name are the same as in the FlexConfig RBS software/FlexDevice hardware For the FlexCard family the naming is based on the following mapping: FlexCard PMC2: Slot 1: Channel 1 and Channel 2 is named as Connector 1A and Connector 1B. For the FlexRay case the channels are mapped to Connector 1AB Slot 2: Channel 3 and Channel 4 is named as Connector 2A and Connector 2B and so on ... FlexCard USB-B FlexRay Connector: FlexRay A+B is named as Connector 1AB CAN Connector:CAN1 is named as Connector 2A, CAN2 is named as Connector 2B, CAN3 is named as Connector 2C.

Fields

NAME	DESCRIPTION
Connector1A	Connector 1A
Connector1AB	Connector 1 A+B. Valid only for a FlexRay Bus.
Connector1B	Connector 1B
Connector1C	Connector 1C
Connector1D	Connector 1D
Connector1E	Connector 1E
Connector1F	Connector 1F
Connector1G	Connector 1G
Connector1H	Connector 1G
Connector2A	Connector 2A

NAME	DESCRIPTION
Connector2AB	Connector 2 A+B. Valid only for a FlexRay Bus.
Connector2B	Connector 2B
Connector2C	Connector 2C
Connector2D	Connector 2D
Connector2E	Connector 2E
Connector2F	Connector 2F
Connector2G	Connector 2G
Connector2H	Connector 2G
Connector3A	Connector 3A
Connector3AB	Connector 3 A+B. Valid only for a FlexRay Bus.
Connector3B	Connector 3B
Connector3C	Connector 3C
Connector3D	Connector 3D
Connector3E	Connector 3E
Connector3F	Connector 3F
Connector3G	Connector 3G

NAME	DESCRIPTION
Connector3H	Connector 3G
Connector4A	Connector 4A
Connector4AB	Connector 4 A+B. Valid only for a FlexRay Bus.
Connector4B	Connector 4B
Connector4C	Connector 4C
Connector4D	Connector 4D
Connector4E	Connector 4E
Connector4F	Connector 4F
Connector4G	Connector 4G
Connector4H	Connector 4G
Connector5A	Connector 5A
Connector5AB	Connector 5 A+B. Valid only for a FlexRay Bus.
Connector5B	Connector 5B
Connector5C	Connector 5C
Connector5D	Connector 5D
Connector5E	Connector 5E

NAME	DESCRIPTION
Connector5F	Connector 5F
Connector5G	Connector 5G
Connector5H	Connector 5G
Unknown	Connector name is not available.

Class DeviceManager

Implementation of IDeviceManager. Handles searching for connected devices and their access state.

Inheritance

Object
DeviceManager

Implements

IDeviceManager

Inherited Members

Object.Equals(Object)
Object.Equals(Object, Object)
Object.GetHashCode()
Object.GetType()
Object.MemberwiseClone()
Object.ReferenceEquals(Object, Object)
Object.ToString()

Namespace: FlexConfig.Sdk.Devices

Assembly: FlexConfig.Sdk.dll

Syntax

```
public class DeviceManager : IDeviceManager
```

Properties

Instance

Gets the one and only instance of IDeviceManager.

Declaration

```
public static IDeviceManager Instance { get; }
```

Property Value

TYPE	DESCRIPTION
IDeviceManager	

Methods

Close(IDevice)

Closes the device.

Declaration

```
public void Close(IDevice device)
```

Parameters

TYPE	NAME	DESCRIPTION
IDevice	device	The device to close.

Exceptions

TYPE	CONDITION
ArgumentNullException	Thrown when variable was null.

GetDevices()

Gets the devices which are available on the system or network.

Declaration

```
public IEnumerable<IDeviceInfo> GetDevices()
```

Returns

TYPE	DESCRIPTION
IEnumerable<IDeviceInfo>	A list of devices which has been found on the system or network.

Exceptions

TYPE	CONDITION
FcBaseApiErrorException	Thrown when error code is returned from fcBase Api.

Open(IDeviceInfo)

Opens the device for exclusive access.

Declaration

```
public IDevice Open(IDeviceInfo deviceInfo)
```

Parameters

TYPE	NAME	DESCRIPTION
IDeviceInfo	deviceInfo	The device which should be opened.

Returns

TYPE	DESCRIPTION
IDevice	The device object if successful. Otherwise null.

Exceptions

TYPE	CONDITION

TYPE	CONDITION
FcBaseApiErrorException	Thrown when error code is returned from fcBase Api.
ArgumentNullException	Thrown when variable was null.
InvalidOperationException	Thrown when the requested device could not be found or is already opened.

SetGlobalConfig(String, String)

Sets global config parameters which allows to customize internal configuration parameters which may change internal behaviour. This method should be called before any method like GetDevices() as the change of global config may have side effects to these methods. For each Parameter the function must be called. If this function is not called, default Parameters are used.

Declaration

```
public void SetGlobalConfig(string key, string value)
```

Parameters

TYPE	NAME	DESCRIPTION
String	key	The name of parameter to change.
String	value	The value of the parameter.

Remarks

Setting "hwComAddress" changes how GetDevices() will enumerate the list. The list will contain all available FlexCards but only the FlexDevice which ip address matches the given hwComAddress. All other FlexDevice will not be listed.

Implements

IDeviceManager

Enum DeviceType

Defines the different device types.

Namespace: FlexConfig.Sdk.Devices

Assembly: FlexConfig.Sdk.dll

Syntax

```
public enum DeviceType
```

Fields

NAME	DESCRIPTION
FlexCardPCIe3	FlexCard PCIe3
FlexCardPMC2	FlexCard PMC2
FlexCardPXIe3	FlexCard PXIe3
FlexCardUSBM	FlexCard USB-M
FlexDeviceL	FlexDevice L
FlexDeviceL2	FlexDevice L2
FlexDeviceS	FlexDevice S
NotListedDevice	Unknown device type.

Enum EthernetMessageType

Enum EthernetMessageType.

Namespace: FlexConfig.Sdk.Devices

Assembly: FlexConfig.Sdk.dll

Syntax

```
public enum EthernetMessageType
```

Fields

NAME	DESCRIPTION
Error	The error
Notification	The notification
Request	The request
RequestNoReturn	The request no return
Response	The response
TpError	The tp error
TpNotification	The tp notification
TpRequest	The tp request
TpRequestNoReturn	The tp request no return
TpResponse	The tp response

Enum EthernetMode

Enum EthernetMode.

Namespace: FlexConfig.Sdk.Devices

Assembly: FlexConfig.Sdk.dll

Syntax

```
public enum EthernetMode
```

Fields

NAME	DESCRIPTION
Mode1000BaseT1Master	BroadR-Reach master 1000BaseT1
Mode1000BaseT1Slave	BroadR-Reach slave 1000BaseT1
Mode1000BaseTXFixed	Ethernet 1000BaseTx
Mode100BaseT1Master	BroadR-Reach master 100BaseT1
Mode100BaseT1Slave	BroadR-Reach slave 100BaseT1
Mode100BaseTXFixed	Ethernet 100BaseTx
Mode10BaseTXFixed	Ethernet 10BaseTx
ModeAuto	Ethernet auto mode

Enum FrameDirection

Defines the possible direction for a message.

Namespace: FlexConfig.Sdk.Devices

Assembly: FlexConfig.Sdk.dll

Syntax

```
public enum FrameDirection
```

Fields

NAME	DESCRIPTION
Receive	The message was received.
Transmit	The message was transmitted.

Interface IBusEvent

Defines the basic interface for events which is generated by a device. The events are received via OnReceive method.

Namespace: FlexConfig.Sdk.Devices

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IBusEvent
```

Properties

BusEventType

Gets the type of bus event.

Declaration

```
BusEventType BusEventType { get; }
```

Property Value

TYPE	DESCRIPTION
BusEventType	

DeviceBusConnector

Gets the bus connector on which the event was received.

Declaration

```
IDeviceBusConnector DeviceBusConnector { get; }
```

Property Value

TYPE	DESCRIPTION
IDeviceBusConnector	The bus connector on which the event was received.

TimeStamp

Gets the timestamp of the event. The timestamp marks the time where the event was received/generated relative to the measurement start.

Declaration

```
TimeSpan TimeStamp { get; }
```

Property Value

TYPE	DESCRIPTION
TimeSpan	The timestamp.

Interface ICanErrorEvent

Defines the interface for keeping information about a CAN/CAN-FD error event.

Inherited Members

IBusEvent.BusEventType
IBusEvent.DeviceBusConnector
IBusEvent.TimeStamp

Namespace: FlexConfig.Sdk.Devices

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface ICanErrorEvent : IStatusEvent, IBusEvent
```

Properties

ErrorType

Gets the Error type.

Declaration

```
BusCanErrorType ErrorType { get; }
```

Property Value

TYPE	DESCRIPTION
BusCanErrorType	

ReceiveErrorCounter

Gets the actual state of the Receive Error Counter. Valid values range from 0 to 127.

Declaration

```
uint ReceiveErrorCounter { get; }
```

Property Value

TYPE	DESCRIPTION
UInt32	

State

Gets the current CAN CC state.

Declaration

```
BusCommunicationControllerState State { get; }
```

Property Value

TYPE	DESCRIPTION
BusCommunicationControllerState	

TransmitErrorCounter

Gets the actual state of the Transmit Error Counter. Values values range 0 to 255.

Declaration

```
uint TransmitErrorCounter { get; }
```

Property Value

TYPE	DESCRIPTION
UInt32	

Interface ICanFrameEvent

Defines a message which was received or transmitted on a CAN or CAN FD bus.

Inherited Members

IFrameEvent.Raw
IFrameEvent.Direction
IFrameEvent.Frame
IFrameEvent.Interpreter
IBusEvent.BusEventType
IBusEvent.DeviceBusConnector
IBusEvent.TimeStamp

Namespace: FlexConfig.Sdk.Devices

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface ICanFrameEvent : IFrameEvent, IBusEvent
```

Properties

BitRateSwitch

Gets a value indicating whether Bit rate switching is activated by the message. This bit is only valid when FdFormat is 'true'.

Declaration

```
bool BitRateSwitch { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	`true` if [bit rate switch]; otherwise, `false`.

ESI

Gets a value indicating whether 'Error state indicator' is set. If it is 'false', the transmitting node is in the error active state. That means the node is allowed to send error frames. If ESI is 'true', the transmitting node is in the error passive state. This bit is only valid when FdFormat is 'true'.

Declaration

```
bool ESI { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	`true` if error passive state; otherwise, `false`.

FdFormat

Gets a value indicating whether the message is in CAN FD format.

Declaration

```
bool FdFormat { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	`true` if [fd format]; otherwise, `false`.

Id

Gets the CAN message identifier which was received or transmitted.

Declaration

```
uint Id { get; }
```

Property Value

TYPE	DESCRIPTION
UInt32	The identifier.

IsExtendedId

Gets a value indicating whether this is an extended Frame. If this flag is 'true' the CAN message is an extended frame. If set to 'false' it is a standard frame.

Declaration

```
bool IsExtendedId { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	`true` if this instance is extended identifier; otherwise, `false`.

MessageLost

Gets a value indicating whether a message was lost. If this flag is 'true' the CAN communication controller has lost a message. If 'false' no message has lost. This flag is only valid with Direction = 'Receive'.

Declaration

```
bool MessageLost { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	`true` if [message lost]; otherwise, `false`.

PayloadLength

Gets the length of the payload of the CAN or CAN FD message.

Declaration

```
uint PayloadLength { get; }
```

Property Value

TYPE	DESCRIPTION
UInt32	The payload length in bytes.

RemoteFrame

Gets a value indicating whether the packet is a remote frame. This flag depends on the parameter Direction. If RemoteFrame is 'true' and Direction is 'Receive', the CAN packet is a remote rx frame. If RemoteFrame is 'true' and Direction is 'Transmit', the CAN packet is a remote tx frame.

Declaration

```
bool RemoteFrame { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	`true` if [remote frame]; otherwise, `false`.

See Also

IFrameEvent

Interface ICommunicationErrorEvent

Defines the interface to identifying communication error events on the bus.

Inherited Members

IBusEvent.BusEventType
IBusEvent.DeviceBusConnector
IBusEvent.TimeStamp

Namespace: FlexConfig.Sdk.Devices

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface ICommunicationErrorEvent : IBusEvent
```

Properties

TypeOfError

Gets the communication error type.

Declaration

```
BusCommunicationErrorType TypeOfError { get; }
```

Property Value

TYPE	DESCRIPTION
BusCommunicationErrorType	

Interface IDevice

Defines the interface for a device which can be used to monitor frames from buses.

Inherited Members

IDisposable.Dispose()

Namespace: FlexConfig.Sdk.Devices

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IDevice : IDisposable
```

Properties

Info

Gets the info about the device.

Declaration

```
IDeviceInfo Info { get; }
```

Property Value

TYPE	DESCRIPTION
IDeviceInfo	

Methods

AddFilter(IDeviceBusConnector, IFrame)

Adds filter so that only the configured frame data will be received. If no filter is configured all bus data will be received.

Declaration

```
bool AddFilter(IDeviceBusConnector busConnector, IFrame frame)
```

Parameters

TYPE	NAME	DESCRIPTION
IDeviceBusConnector	busConnector	The bus connector on which the filter should be applied.
IFrame	frame	The frame which should be received.

Returns

TYPE	DESCRIPTION
Boolean	True if the filter was added. Otherwise false.

AddFilter(IDeviceBusConnector, IPdu)

Adds filter so that only the configured PDU data will be received. If no filter is configured all bus data will be received.

Adds filter so that only the configured PDU data will be received. If no filter is configured all bus data will be received.

Declaration

```
bool AddFilter(IDeviceBusConnector busConnector, IPdu pdu)
```

Parameters

TYPE	NAME	DESCRIPTION
IDeviceBusConnector	busConnector	The bus connector on which the filter should be applied.
IPdu	pdu	The pdu which should be received.

Returns

TYPE	DESCRIPTION
Boolean	True if the filter was added. Otherwise false.

AddFilter(IDeviceBusConnector, ITriggering)

Adds filter so that only the configured frame data will be received. If no filter is configured all bus data will be received.

Declaration

```
bool AddFilter(IDeviceBusConnector busConnector, ITriggering triggering)
```

Parameters

TYPE	NAME	DESCRIPTION
IDeviceBusConnector	busConnector	The bus connector on which the filter should be applied.
ITriggering	triggering	The bus triggering which should used to configure the receive filter.

Returns

TYPE	DESCRIPTION
Boolean	True if the filter was added. Otherwise false.

AddFilter(IDeviceBusConnector, UInt16[], IPAddress, IPAddress, Nullable<UInt16>, Nullable<UInt16>, PhysicalAddress, PhysicalAddress, Nullable<UInt16>)

Sets the ethernet filter. If needed, this method should be called after the 'SetEthernetConfiguration' method.

Declaration

```
bool AddFilter(IDeviceBusConnector busConnector, ushort[] vlanTags = null, IPAddress sourceIpV4Adress = null,
IPAddress destinationIpV4Adress = null, ushort? sourcePort = null, ushort? destinationPort = null,
PhysicalAddress sourcePhysicalAddress = null, PhysicalAddress destinationPhysicalAddress = null, ushort?
etherType = null)
```

Parameters

TYPE	NAME	DESCRIPTION
IDeviceBusConnector	busConnector	The bus connector.
UInt16[]	vlanTags	The vlan tags.
IPAddress	sourceIpV4Adress	The source ip v4 adress.
IPAddress	destinationIpV4Adress	The destination ip v4 adress.
Nullable<UInt16>	sourcePort	The source port.
Nullable<UInt16>	destinationPort	The destination port.
PhysicalAddress	sourcePhysicalAddress	The source physical address.
PhysicalAddress	destinationPhysicalAddress	The destination physical address.
Nullable<UInt16>	etherType	Type of the ether.

Returns

TYPE	DESCRIPTION
Boolean	`true` if the filter is set, `false` otherwise.

ClearFilter()

Removes all filter. All bus data will be received.

Declaration

```
bool ClearFilter()
```

Returns

TYPE	DESCRIPTION
Boolean	True if the filter has been cleared. Otherwise false.

Connect(IDeviceBusConnector, IBus, String)

Connects the given bus to the connector of this device. e.g. FlexCard part: Configures the CCs according bus information via fcAPI Chi .... Marks this connector so that data can be received.

Declaration

```
bool Connect(IDeviceBusConnector busConnector, IBus bus, string busName = null)
```

Parameters

TYPE	NAME	DESCRIPTION
IDeviceBusConnector	busConnector	The bus connector of the device.
IBus	bus	The bus which should be disconnected.
String	busName	The bus name.

Returns

TYPE	DESCRIPTION
Boolean	True if successful. Otherwise false.

Exceptions

TYPE	CONDITION
ArgumentNullException	Thrown when variable was null.
InvalidOperationException	Thrown when the bus type does not matches the bus connector type.

Disconnect(IDeviceBusConnector, IBus)

Disconnect the bus from the connector of this device.

Declaration

```
bool Disconnect(IDeviceBusConnector busConnector, IBus bus)
```

Parameters

TYPE	NAME	DESCRIPTION
IDeviceBusConnector	busConnector	The bus connector of the device.
IBus	bus	The bus which should be connected.

Returns

TYPE	DESCRIPTION
Boolean	True if successful. Otherwise false.

Exceptions

TYPE	CONDITION
FcBaseApiErrorException	Thrown when error code is returned from fcBase Api.
ArgumentOutOfRangeException	Thrown when variable was not listed in switch case.

SetEthernetConfiguration(IDeviceBusConnector, EthernetMode, Int16[])

Sets the ethernet configuration.

Declaration

```
bool SetEthernetConfiguration(IDeviceBusConnector busConnector, EthernetMode ethernetMode, short[] vlanTags)
```

Parameters

TYPE	NAME	DESCRIPTION
IDeviceBusConnector	busConnector	The bus connector on which the filter should be applied.
EthernetMode	ethernetMode	The ethernet mode of the bus connector.
Int16[]	vlanTags	The V-LAN tags which should be considered by the communication.

Returns

TYPE	DESCRIPTION
Boolean	True if the configuration was successful. Otherwise false.

StartMeasurement()

Starts the measurement on the connected buses Connect(IDeviceBusConnector, IBus, String).

Declaration

```
void StartMeasurement()
```

Exceptions

TYPE	CONDITION
FcBaseApiErrorException	Thrown when error code is returned from fcBase Api.
ArgumentOutOfRangeException	Thrown when variable was not listed in switch case.

StopMeasurement()

Stops the measurement on the connected buses.

Declaration

```
void StopMeasurement()
```

Transmit(IDeviceBusConnector, ITransmitableFrame)

Sends a transmitable frame from the bus.

Declaration

```
void Transmit(IDeviceBusConnector busConnector, ITransmitableFrame transmitable)
```

Parameters

TYPE	NAME	DESCRIPTION
IDeviceBusConnector	busConnector	The busconnector from which will be sended.
ITransmitableFrame	transmitable	The transmitable which will be sended.

Events

OnConnectionLost

The event which is used to receive the information if the network connection to the device is lost.

Declaration

```
event EventHandler<EventArgs> OnConnectionLost
```

Event Type

TYPE	DESCRIPTION
EventHandler<EventArgs>	

OnReceive

The event which is used to receive the events from the buses.

Declaration

```
event EventHandler<BusEventEventArgs> OnReceive
```

Event Type

TYPE	DESCRIPTION
EventHandler<BusEventEventArgs>	

Interface IDeviceBusConnector

Gets information about a device bus connector, e.g. which bus type it supports, on which physical port the bus is available.

Namespace: FlexConfig.Sdk.Devices

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IDeviceBusConnector
```

Properties

BusName

Gets the name of the bus.

Declaration

```
string BusName { get; }
```

Property Value

TYPE	DESCRIPTION
String	The name of the bus.

Examples

FR1, CAN5. Values can be set by the user.

BusType

Gets the bus type.

Declaration

```
BusType BusType { get; }
```

Property Value

TYPE	DESCRIPTION
BusType	

CcType

Gets the type of the communication controller.

Declaration

```
CcType CcType { get; }
```

Property Value

TYPE	DESCRIPTION
CcType	

Index

Gets the communication controller index of the bus connector.

Declaration

```
Cc Index { get; }
```

Property Value

TYPE	DESCRIPTION
Ebel.Hardware.FlexCard.Cc	

IsBeingMonitored

Gets a value indicating whether device bus connector is being monitored.

Declaration

```
bool IsBeingMonitored { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

IsConfigured

Gets a value indicating whether device bus connector is configured.

Declaration

```
bool IsConfigured { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

Name

Gets the name of the device bus connector. The name specifies on which physical connector the bus is available.

Declaration

```
DeviceBusConnectorName Name { get; }
```

Property Value

TYPE	DESCRIPTION
DeviceBusConnectorName	

Interface IDeviceInfo

Defines the information from a device. This includes the type of device and which buses are supported by the device.

Namespace: FlexConfig.Sdk.Devices

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IDeviceInfo
```

Properties

BusConnectors

Gets the supported buses.

Declaration

```
IEnumerable<IDeviceBusConnector> BusConnectors { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<IDeviceBusConnector>	

DeviceType

Gets the device type.

Declaration

```
DeviceType DeviceType { get; }
```

Property Value

TYPE	DESCRIPTION
DeviceType	

SerialNumber

Gets the serial number of the device.

Declaration

```
string SerialNumber { get; }
```

Property Value

TYPE	DESCRIPTION
String	

Interface IDeviceManager

Handles the enumeration, opening and closing devices.

Namespace: FlexConfig.Sdk.Devices

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IDeviceManager
```

Methods

Close(IDevice)

Closes the device.

Declaration

```
void Close(IDevice device)
```

Parameters

TYPE	NAME	DESCRIPTION
IDevice	device	The device to close.

Exceptions

TYPE	CONDITION
ArgumentNullException	Thrown when variable was null.

GetDevices()

Gets the devices which are available on the system or network.

Declaration

```
IEnumerable<IDeviceInfo> GetDevices()
```

Returns

TYPE	DESCRIPTION
IEnumerable<IDeviceInfo>	A list of devices which has been found on the system or network.

Exceptions

TYPE	CONDITION
FcBaseApiErrorException	Thrown when error code is returned from fcBase Api.

Open(IDeviceInfo)

Opens the device for exclusive access.

Declaration

```
IDevice Open(IDeviceInfo deviceInfo)
```

Parameters

TYPE	NAME	DESCRIPTION
IDeviceInfo	deviceInfo	The device which should be opened.

Returns

TYPE	DESCRIPTION
IDevice	The device object if successful. Otherwise null.

Exceptions

TYPE	CONDITION
FcBaseApiErrorException	Thrown when error code is returned from fcBase Api.
ArgumentNullException	Thrown when variable was null.
InvalidOperationException	Thrown when the requested device could not be found or is already opened.

SetGlobalConfig(String, String)

Sets global config parameters which allows to customize internal configuration parameters which may change internal behaviour. This method should be called before any method like GetDevices() as the change of global config may have side effects to these methods. For each Parameter the function must be called. If this function is not called, default Parameters are used.

Declaration

```
void SetGlobalConfig(string key, string value)
```

Parameters

TYPE	NAME	DESCRIPTION
String	key	The name of parameter to change.
String	value	The value of the parameter.

Remarks

Setting "hwComAddress" changes how GetDevices() will enumerate the list. The list will contain all available FlexCards but only

the FlexDevice which ip address matches the given hwComAddress. All other FlexDevice will not be listed.

Examples

SetGlobalConfig("hwComAddress", "192.168.1.15"); where 192.168.1.15 is the ip of the device to be opened.

Interface IEthernetEvent

Defines the interface for keeping information about decoded ethernet package.

Namespace: FlexConfig.Sdk.Devices

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IEthernetEvent
```

Properties

DestinationHardwareAddress

Gets the destination hardware address.

Declaration

```
PhysicalAddress DestinationHardwareAddress { get; }
```

Property Value

TYPE	DESCRIPTION
PhysicalAddress	The destination hardware address.

DestinationIpAddress

Gets the target ip address.

Declaration

```
IPAddress DestinationIpAddress { get; }
```

Property Value

TYPE	DESCRIPTION
IPAddress	The target ip address.

DestinationPort

Gets the target port.

Declaration

```
ushort DestinationPort { get; }
```

Property Value

TYPE	DESCRIPTION
UInt16	The target port.

IpVersion

Gets the ip version.

Declaration

```
IpVersion IpVersion { get; }
```

Property Value

TYPE	DESCRIPTION
IpVersion	The ip version.

Protocol

Gets the protocol.

Declaration

```
TransportProtocol Protocol { get; }
```

Property Value

TYPE	DESCRIPTION
TransportProtocol	The protocol.

SourceHardwareAddress

Gets the source hardware address.

Declaration

```
PhysicalAddress SourceHardwareAddress { get; }
```

Property Value

TYPE	DESCRIPTION
PhysicalAddress	The source hardware address.

SourceIpAddress

Gets the source ip address.

Declaration

```
IPAddress SourceIpAddress { get; }
```

Property Value

TYPE	DESCRIPTION
IPAddress	The source ip address.

SourcePort

Gets the source port.

Declaration

```
ushort SourcePort { get; }
```

Property Value

TYPE	DESCRIPTION
UInt16	The source port.

VlanIdentifier

Gets the vlan identifier.

Declaration

```
ushort VlanIdentifier { get; }
```

Property Value

TYPE	DESCRIPTION
UInt16	The vlan identifier.

Interface IEthernetFrameEvent

Defines the interface for an ethernet frame event which is not related with a service element. Implements the IFrameEvent Implements the IEthernetEvent.

Inherited Members

IFrameEvent.Raw
IFrameEvent.Direction
IFrameEvent.Frame
IFrameEvent.Interpreter
IBusEvent.BusEventType
IBusEvent.DeviceBusConnector
IBusEvent.TimeStamp
IEthernetEvent.SourceHardwareAddress
IEthernetEvent.DestinationHardwareAddress
IEthernetEvent.VlanIdentifier
IEthernetEvent.IpVersion
IEthernetEvent.SourceIpAddress
IEthernetEvent.DestinationIpAddress
IEthernetEvent.Protocol
IEthernetEvent.SourcePort
IEthernetEvent.DestinationPort

Namespace: FlexConfig.Sdk.Devices

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IEthernetFrameEvent : IFrameEvent, IBusEvent, IEthernetEvent
```

Properties

HeaderId

Gets the header identifier.

Declaration

```
uint HeaderId { get; }
```

Property Value

TYPE	DESCRIPTION
UInt32	The header identifier.

See Also

IFrameEvent
IEthernetEvent

Interface IEthernetServiceEvent

Defines the interface for an ethernet service elements such as events, methods and fields. Implements the IEthernetEvent.

Inherited Members

IEthernetEvent.SourceHardwareAddress
IEthernetEvent.DestinationHardwareAddress
IEthernetEvent.VlanIdentifier
IEthernetEvent.IpVersion
IEthernetEvent.SourceIpAddress
IEthernetEvent.DestinationIpAddress
IEthernetEvent.Protocol
IEthernetEvent.SourcePort
IEthernetEvent.DestinationPort
IBusEvent.BusEventType
IBusEvent.DeviceBusConnector
IBusEvent.TimeStamp

Namespace: FlexConfig.Sdk.Devices

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IEthernetServiceEvent : IEthernetEvent, IBusEvent
```

Properties

DataTypeDeclarations

Gets the data type declarations.

Declaration

```
IEnumerable<IDataTypeDeclaration> DataTypeDeclarations { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<IDataTypeDeclaration>	The data type declarations.

Interpreters

Gets the interpreters.

Declaration

```
IEnumerable<IDataTypeInterpreter> Interpreters { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<IDataTypeInterpreter>	The interpreters.

MessageType

Gets the type of the message.

Declaration

```
EthernetMessageType MessageType { get; }
```

Property Value

TYPE	DESCRIPTION
EthernetMessageType	The type of the message.

MethodId

Gets the method identifier.

Declaration

```
ushort MethodId { get; }
```

Property Value

TYPE	DESCRIPTION
UInt16	The method identifier.

ServiceElementName

Gets the name of the service element.

Declaration

```
string ServiceElementName { get; }
```

Property Value

TYPE	DESCRIPTION
String	The name of the service element.

ServiceElementType

Gets the type of the service element.

Declaration

```
ServiceElementType ServiceElementType { get; }
```

Property Value

TYPE	DESCRIPTION
ServiceElementType	The type of the service element.

ServiceInstanceId

Gets the provided service instance identifier.

Declaration

```
ushort ServiceInstanceId { get; }
```

Property Value

TYPE	DESCRIPTION
UInt16	The service instance identifier.

ServiceInterface

Gets the service interface.

Declaration

```
IServiceInterface ServiceInterface { get; }
```

Property Value

TYPE	DESCRIPTION
IServiceInterface	The service interface.

See Also

IEthernetEvent

Interface IFlexRayErrorEvent

Defines the interface for keeping information about a FlexRay error event.

Inherited Members

IBusEvent.BusEventType
IBusEvent.DeviceBusConnector
IBusEvent.TimeStamp

Namespace: FlexConfig.Sdk.Devices

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IFlexRayErrorEvent : IStatusEvent, IBusEvent
```

Properties

ErrorType

Gets the error.

Declaration

```
BusFlexRayErrorType ErrorType { get; }
```

Property Value

TYPE	DESCRIPTION
BusFlexRayErrorType	

Interface IFlexRayFrameEvent

Defines a message which was received or transmitted on a FlexRay bus.

Inherited Members

IFrameEvent.Raw
IFrameEvent.Direction
IFrameEvent.Frame
IFrameEvent.Interpreter
IBusEvent.BusEventType
IBusEvent.DeviceBusConnector
IBusEvent.TimeStamp

Namespace: FlexConfig.Sdk.Devices

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IFlexRayFrameEvent : IFrameEvent, IBusEvent
```

Properties

AsyncMode

Gets a value indicating whether [asynchronous mode] is active. If the packet was generated by the asynchronous debug Mode, this parameter is set to 'true'.

Declaration

```
bool AsyncMode { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	`true` if [asynchronous mode]; otherwise, `false`.

Channel

Gets the channel (A or B) on which the frame was received.

Declaration

```
FlexRayChannel Channel { get; }
```

Property Value

TYPE	DESCRIPTION
FlexRayChannel	The channel.

ContentError

Gets a value indicating whether the message has a content error. If a content error was observed, this parameter is set to 'true'.

Declaration

```
bool ContentError { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	`true` if [content error]; otherwise, `false`.

CycleCount

Gets the cycle in which the frame was received.

Declaration

```
uint CycleCount { get; }
```

Property Value

TYPE	DESCRIPTION
UInt32	The cycle count.

FrameCRC

Gets the frame CRC. If the packet was generated by the asynchronous debug Mode, the FrameCRC contains the cyclic redundancy check code computed over a complete frame. In synchronous monitoring Mode, this parameter is not set.

Declaration

```
uint FrameCRC { get; }
```

Property Value

TYPE	DESCRIPTION
UInt32	The frame CRC.

HeaderCRC

Gets the header CRC. The header CRC contains the cyclic redundancy check code is computed over the sync frame indicator, the start up frame indicator, the frame id and the payload length.

Declaration

```
uint HeaderCRC { get; }
```

Property Value

TYPE	DESCRIPTION
UInt32	The header CRC.

Id

Gets the frame id, which defines the slot in which the frame was transmitted or received.

Declaration

```
uint Id { get; }
```

Property Value

TYPE	DESCRIPTION
UInt32	The slot identifier.

IsNullFrame

Gets a value indicating whether this instance is a null frame (=true) or have valid data (=false).

Declaration

```
bool IsNullFrame { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	`true` if this instance is null frame; otherwise, `false`.

PayloadLength

Gets the length of the payload (byte length).

Declaration

```
uint PayloadLength { get; }
```

Property Value

TYPE	DESCRIPTION
UInt32	The length of the payload.

Pp

Gets a value indicating whether the payload preamble indicator is set, which defines whether or not an optional vector is contained within the payload segment of the frame transmitted. In the static segment, it indicates the presence of a network management vector at the beginning of the payload. In the dynamic segment it indicates the presence of a message id at the beginning of the payload.

Declaration

```
bool Pp { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	

ReservedBit

Gets a value indicating whether reserved bit is set.

Declaration

```
bool ReservedBit { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	`true` if [reserved bit]; otherwise, `false`.

SlotBoundaryViolation

Gets a value indicating whether a [slot boundary violation] is observed. If a slot boundary violation was observed, this parameter is set to 'true'.

Declaration

```
bool SlotBoundaryViolation { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	`true` if [slot boundary violation]; otherwise, `false`.

Startup

Gets a value indicating whether the frame is a start up frame (=true) or not (=false).

Declaration

```
bool Startup { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	`true` if startup; otherwise, `false`.

Sync

Gets a value indicating whether the frame is a sync frame (true) or not (false).

Declaration

```
bool Sync { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	`true` if synchronize; otherwise, `false`.

SyntaxError

Gets a value indicating whether message has a syntax error. If a syntax error was observed, this parameter is set to 'true'.

Declaration

```
bool SyntaxError { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	`true` if [syntax error]; otherwise, `false`.

ValidFrame

Gets a value indicating whether [valid frame]. If a valid frame was received, this parameter is set to 'true'.

Declaration

```
bool ValidFrame { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	`true` if [valid frame]; otherwise, `false`.

See Also

IFrameEvent

Interface IFlexRayStatusEvent

Defines the interface for keeping information about a FlexRay status event.

Inherited Members

IBusEvent.BusEventType
IBusEvent.DeviceBusConnector
IBusEvent.TimeStamp

Namespace: FlexConfig.Sdk.Devices

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IFlexRayStatusEvent : IStatusEvent, IBusEvent
```

Properties

CycleCount

Gets the cycle in which the status has changed.

Declaration

```
uint CycleCount { get; }
```

Property Value

TYPE	DESCRIPTION
UInt32	

Status

Gets the bus status flag.

Declaration

```
BusStatusFlag Status { get; }
```

Property Value

TYPE	DESCRIPTION
BusStatusFlag	The flag.

WakeupInfo

Gets the wake up status.

Declaration

```
BusFlexRayWakeupStatus WakeupInfo { get; }
```

Property Value

TYPE	DESCRIPTION
BusFlexRayWakeupStatus	

Interface IFlexRaySyncInformationEvent

Contains information about a FlexRay synchronization.

Inherited Members

IBusEvent.BusEventType
IBusEvent.DeviceBusConnector
IBusEvent.TimeStamp

Namespace: FlexConfig.Sdk.Devices

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IFlexRaySyncInformationEvent : IStatusEvent, IBusEvent
```

Properties

ClockCorrectionFailedCounter

Gets the correction failed counter (FlexRay Protocol Specification: vClockCorrectionFailed).

Declaration

```
uint ClockCorrectionFailedCounter { get; }
```

Property Value

TYPE	DESCRIPTION
UInt32	

CurrentCycle

Gets the current FlexRay cycle.

Declaration

```
uint CurrentCycle { get; }
```

Property Value

TYPE	DESCRIPTION
UInt32	

OffsetCorrection

Gets the offset correction value (two' complement). Indicates the number of microticks that are added to the offset correction segment of the network idle time.

Declaration

```
uint OffsetCorrection { get; }
```

Property Value

TYPE	DESCRIPTION

TYPE	DESCRIPTION
UInt32	

PassiveToActiveCount

Gets the passiv to active value (FlexRay Protocol Specification: vAllowPassiveToActive).

Declaration

```
uint PassiveToActiveCount { get; }
```

Property Value

TYPE	DESCRIPTION
UInt32	

RateCorrection

Gets the rate correction value (complement on two). Indicates by how many microticks the node's cycle length should be changed.

Declaration

```
uint RateCorrection { get; }
```

Property Value

TYPE	DESCRIPTION
UInt32	

Interface IFrameEvent

Defines the interface for a frame event.

Inherited Members

IBusEvent.BusEventType
IBusEvent.DeviceBusConnector
IBusEvent.TimeStamp

Namespace: FlexConfig.Sdk.Devices

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IFrameEvent : IBusEvent
```

Properties

Direction

Gets the direction (receive or transmit) of the message.

Declaration

```
FrameDirection Direction { get; }
```

Property Value

TYPE	DESCRIPTION
FrameDirection	The direction.

Frame

Gets the frame object, which is defined in the database.

Declaration

```
IFrame Frame { get; }
```

Property Value

TYPE	DESCRIPTION
IFrame	The database frame.

Interpreter

Gets the interpreter for converting the frame information into something readable.

Declaration

```
IFrameInterpreter Interpreter { get; }
```

Property Value

TYPE	DESCRIPTION
IFrameInterpreter	

Raw

Gets the raw bytes of the message. Use the 'ToArray' method in order to get a byte array.

Declaration

```
ReadOnlySpan<byte> Raw { get; }
```

Property Value

TYPE	DESCRIPTION
ReadOnlySpan<Byte>	

Interface IMultiplexerSegment

The segment of a multiplexer PDU.

Namespace: FlexConfig.Sdk.Devices

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IMultiplexerSegment
```

Properties

BitLength

Gets the bit length.

Declaration

```
int BitLength { get; }
```

Property Value

TYPE	DESCRIPTION
Int32	

BitPosition

Gets the bit position.

Declaration

```
int BitPosition { get; }
```

Property Value

TYPE	DESCRIPTION
Int32	

ByteOrder

Gets the byte order.

Declaration

```
ByteOrder ByteOrder { get; }
```

Property Value

TYPE	DESCRIPTION
ByteOrder	

SegmentType

Gets the segment type.

Declaration

```
MultiplexerSegmentType SegmentType { get; }
```

Property Value

TYPE	DESCRIPTION
MultiplexerSegmentType	

Interface IStatusEvent

Defines the interface to identifying status events from the bus.

Inherited Members

IBusEvent.BusEventType

IBusEvent.DeviceBusConnector

IBusEvent.TimeStamp

Namespace: FlexConfig.Sdk.Devices

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IStatusEvent : IBusEvent
```

Interface ISwitchCodeSignal

The switch code of a multiplexed PDU.

Namespace: FlexConfig.Sdk.Devices

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface ISwitchCodeSignal
```

Properties

Item[Int32]

Gets or sets a byte in the switch code byte array.

Declaration

```
byte this[int index] { get; set; }
```

Parameters

TYPE	NAME	DESCRIPTION
Int32	index	The position of the byte to be changed.

Property Value

TYPE	DESCRIPTION
Byte	The byte at the named index position.

RawSwitchCode

Gets or sets the raw switch code.

Declaration

```
byte[] RawSwitchCode { get; set; }
```

Property Value

TYPE	DESCRIPTION
Byte[]	

SwitchCode

Gets or sets the switch code.

Declaration

```
ulong SwitchCode { get; set; }
```

Property Value

TYPE	DESCRIPTION
UInt64	

Interface ITransmitableContainerPdu

The container PDU.

Namespace: FlexConfig.Sdk.Devices

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface ITransmitableContainerPdu
```

Properties

Item[Int32]

Gets the byte of the container PDU.

Declaration

```
byte this[int index] { get; }
```

Parameters

TYPE	NAME	DESCRIPTION
Int32	index	The position of the byte in the payload.

Property Value

TYPE	DESCRIPTION
Byte	

Remarks

Will be updated automatically, if frame payload or signal value have been changed before. Changing the returned byte value via the get method won't change the payload.

MultiplexerPdus

Gets all added contained PDUs in this container PDU.

Declaration

```
List<ITransmitableMultiplexerPdu> MultiplexerPdus { get; }
```

Property Value

TYPE	DESCRIPTION
List<ITransmitableMultiplexerPdu>	

PossibleMultiplexerPdus

Gets all addable contained PDUs for this container PDU. Returns an empty list if no multiplexer PDUs were configured for this container PDU.

Declaration

```
IEnumerable<ContainedMultiplexerPdu> PossibleMultiplexerPdus { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<ContainedMultiplexerPdu>	

PossibleStandardPdus

Gets all addable contained PDUs for this container PDU. Returns an empty list if no standard PDUs were configured for this container PDU.

Declaration

```
IEnumerable<ContainedStandardPdu> PossibleStandardPdus { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<ContainedStandardPdu>	

Specification

Gets the container PDU specification.

Declaration

```
IContainerPduSpecification Specification { get; }
```

Property Value

TYPE	DESCRIPTION
IContainerPduSpecification	

StandardPdus

Gets all added contained PDUs in this container PDU.

Declaration

```
List<ITransmitablePdu> StandardPdus { get; }
```

Property Value

TYPE	DESCRIPTION
List<ITransmitablePdu>	

Methods

AddPdu(ITransmitableMultiplexerPdu)

Adds a multiplexer PDU to this container PDU.

Declaration

```
void AddPdu(ITransmitableMultiplexerPdu pdu)
```

Parameters

TYPE	NAME	DESCRIPTION
ITransmitableMultiplexerPdu	pdu	The pdu to be added/overwritten.

AddPdu(ITransmitablePdu)

Adds a standard PDU to this container PDU.

Declaration

```
void AddPdu(ITransmitablePdu pdu)
```

Parameters

TYPE	NAME	DESCRIPTION
ITransmitablePdu	pdu	The pdu to be added/overwritten.

CreateContainedPdu(ContainedMultiplexerPdu)

Creates a multiplexer PDU which can be added to the container PDU.

Declaration

```
ITransmitableMultiplexerPdu CreateContainedPdu(ContainedMultiplexerPdu multiplexerPdu)
```

Parameters

TYPE	NAME	DESCRIPTION
ContainedMultiplexerPdu	multiplexerPdu	The contained multiplexer PDU information.

Returns

TYPE	DESCRIPTION
ITransmitableMultiplexerPdu	The contained multiplexer PDU object.

CreateContainedPdu(ContainedStandardPdu)

Creates a standard PDU which can be added to the container PDU.

Declaration

```
ITransmitablePdu CreateContainedPdu(ContainedStandardPdu standardPdu)
```

Parameters

TYPE	NAME	DESCRIPTION
ContainedStandardPdu	standardPdu	The contained standard PDU information.

Returns

TYPE	DESCRIPTION
ITransmitablePdu	The contained standard PDU object.

GetPayload()

Gets the payload of the container PDU.

Declaration

```
byte[] GetPayload()
```

Returns

TYPE	DESCRIPTION
Byte[]	The container PDU payload.

Remarks

Changes on this byte array won't change the PDU payload.

Interface ITransmitableFrame

The frame to be transmitted.

Namespace: FlexConfig.Sdk.Devices

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface ITransmitableFrame
```

Properties

ContainerPdus

Gets the container PDUs. Returns an empty list if no container PDUs were configured for this frame.

Declaration

```
IEnumerable<ITransmitableContainerPdu> ContainerPdus { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<ITransmitableContainerPdu>	

Item[Int32]

Gets or sets the byte of the frame.

Declaration

```
byte this[int index] { get; set; }
```

Parameters

TYPE	NAME	DESCRIPTION
Int32	index	The position of the byte in the frame payload.

Property Value

TYPE	DESCRIPTION
Byte	

Remarks

Will be updated automatically, if PDU payload or signal value have been changed before. Changing the returned byte value via the get method won't change the PDU payload. Use the Set-method to change the payload.

MultiplexerPdus

Gets the multiplexer PDUs. Returns an empty list if no multiplexer PDUs were configured for this frame.

Declaration

```
IEnumerable<ITransmitableMultiplexerPdu> MultiplexerPdus { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<ITransmitableMultiplexerPdu>	

Pdus

Gets the PDUs. Returns an empty list if no standard PDUs were configured for this frame.

Declaration

```
IEnumerable<ITransmitablePdu> Pdus { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<ITransmitablePdu>	

Specification

Gets the specification of the content of this frame.

Declaration

```
IFrameSpecification Specification { get; }
```

Property Value

TYPE	DESCRIPTION
IFrameSpecification	

Triggering

Gets the information on which timing this element is transmitted or received.

Declaration

```
ITriggering Triggering { get; }
```

Property Value

TYPE	DESCRIPTION
ITriggering	

Methods

GetPayload()

Gets the payload of the frame.

Declaration

```
byte[] GetPayload()
```

Returns

TYPE	DESCRIPTION
Byte[]	The frame payload.

Remarks

Changes on this byte array won't change the frame payload. Use the Set-methods to change the payload.

SetPayload(Byte[])

Sets the payload of the frame.

Declaration

```
void SetPayload(byte[] payload)
```

Parameters

TYPE	NAME	DESCRIPTION
Byte[]	payload	The new frame Payload.

Remarks

Updates PDU payload and signal value. Will be updated automatically, if PDU payload or signal value have been changed before.

Interface ITransmitableMultiplexerPdu

The multiplexer PDU.

Namespace: FlexConfig.Sdk.Devices

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface ITransmitableMultiplexerPdu
```

Properties

Item[Int32]

Gets or sets the byte of the container PDU.

Declaration

```
byte this[int index] { get; set; }
```

Parameters

TYPE	NAME	DESCRIPTION
Int32	index	The position of the byte in the payload.

Property Value

TYPE	DESCRIPTION
Byte	

Remarks

Will be updated automatically, if frame payload or signal value have been changed before. Changing the returned byte value via the get method won't change the payload. Use the Set-method to change the payload.

Specification

Gets the Multiplexer PDU specification.

Declaration

```
IMultiplexerPduSpecification Specification { get; }
```

Property Value

TYPE	DESCRIPTION
IMultiplexerPduSpecification	

StaticPdu

Gets the PDU in the static part.

Declaration

```
ITransmitableStaticPdu StaticPdu { get; }
```

Property Value

TYPE	DESCRIPTION
ITransmitableStaticPdu	

SwitchCodeSignal

Gets the switch code / selector field code.

Declaration

```
ISwitchCodeSignal SwitchCodeSignal { get; }
```

Property Value

TYPE	DESCRIPTION
ISwitchCodeSignal	

SwitchedPdu

Gets the PDU in the dynamic part.

Declaration

```
ITransmitableSwitchedPdu SwitchedPdu { get; }
```

Property Value

TYPE	DESCRIPTION
ITransmitableSwitchedPdu	

Methods

GetPayload()

Gets the payload of the container PDU.

Declaration

```
byte[] GetPayload()
```

Returns

TYPE	DESCRIPTION
Byte[]	The container PDU payload.

Remarks

Changes on this byte array won't change the PDU payload. Use the Set-methods to change the payload.

SetPayload(Byte[])

Sets the payload of the container PDU.

Declaration

```
void SetPayload(byte[] newPayload)
```

Parameters

TYPE	NAME	DESCRIPTION
Byte[]	newPayload	The new container PDU Payload.

Interface ITransmitablePdu

The PDU to be transmitted with its frame.

Namespace: FlexConfig.Sdk.Devices

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface ITransmitablePdu
```

Properties

Item[Int32]

Gets or sets the byte of the PDU.

Declaration

```
byte this[int index] { get; set; }
```

Parameters

TYPE	NAME	DESCRIPTION
Int32	index	The position of the byte in the PDU payload.

Property Value

TYPE	DESCRIPTION
Byte	

Remarks

Will be updated automatically, if frame payload or signal value have been changed before. Changing the returned byte value via the get method won't change the PDU payload. Use the Set-method to change the payload.

Signals

Gets the signals.

Declaration

```
IEnumerable<ITransmitableSignal> Signals { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<ITransmitableSignal>	

Specification

Gets the specification of the content of this PDU.

Declaration

```
IPduSpecification Specification { get; }
```

Property Value

TYPE	DESCRIPTION
IPduSpecification	

Methods

GetPayload()

Gets the payload of the PDU.

Declaration

```
byte[] GetPayload()
```

Returns

TYPE	DESCRIPTION
Byte[]	The PDU payload.

Remarks

Changes on this byte array won't change the PDU payload. Use the Set-methods to change the payload.

SetPayload(Byte[])

Sets the payload of the PDU.

Declaration

```
void SetPayload(byte[] payload)
```

Parameters

TYPE	NAME	DESCRIPTION
Byte[]	payload	The new pdu Payload.

Remarks

Updates frame payload and signal value. Will be updated automatically, if frame payload or signal value have been changed before.

Interface ITransmitableSignal

The signal to be transmitted with its PDU.

Inherited Members

ISignalValue.RawValue
ISignalValue.InternalValue
ISignalValue.InterpretedValue
ISignalValue.TextValue
ISignalValue.IsRawDefined
ISignalValue.IsInternalDefined
ISignalValue.IsPhysicalDefined
ISignalValue.IsTextDefined
ISignalValue.Unit

Namespace: FlexConfig.Sdk.Devices

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface ITransmitableSignal : ISignalValue
```

Properties

Item[Int32]

Gets or sets the byte of the signal.

Declaration

```
byte this[int index] { get; set; }
```

Parameters

TYPE	NAME	DESCRIPTION
Int32	index	The position of the byte in the PDU payload.

Property Value

TYPE	DESCRIPTION
Byte	

Remarks

Changing the returned byte value via the get method won't change the signal values. Use the Set-method to change the values.

If signal is big endian, the correct calculated byte array will be returned and doesn't need to be reversed. Also setting a byte in a big endian signal will change the signal raw value at the correct calculated position.

Specification

Gets the specification of the content of this signal.

Declaration

```
ISignalSpecification Specification { get; }
```

Property Value

TYPE	DESCRIPTION
ISignalSpecification	

Interface ITransmitableStaticPdu

The PDU in the static part of the multiplexer PDU.

Inherited Members

ITransmitablePdu.Specification
ITransmitablePdu.Signals
ITransmitablePdu.Item[Int32]
ITransmitablePdu.SetPayload(Byte[])
ITransmitablePdu.GetPayload()

Namespace: FlexConfig.Sdk.Devices

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface ITransmitableStaticPdu : ITransmitablePdu
```

Interface ITransmitableSwitchedPdu

The switched PDU in a dynamic segment.

Inherited Members

ITransmitablePdu.Specification
ITransmitablePdu.Signals
ITransmitablePdu.Item[Int32]
ITransmitablePdu.SetPayload(Byte[])
ITransmitablePdu.GetPayload()

Namespace: FlexConfig.Sdk.Devices

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface ITransmitableSwitchedPdu : ITransmitablePdu
```

Properties

SwitchCode

Gets the switch code with which this PDU is being used.

Declaration

```
ulong SwitchCode { get; }
```

Property Value

TYPE	DESCRIPTION
UInt64	The switch code.

Enum MultiplexerSegmentType

Defines the possible segment types for a segment in a multiplexer PDU.

Namespace: FlexConfig.Sdk.Devices

Assembly: FlexConfig.Sdk.dll

Syntax

```
public enum MultiplexerSegmentType
```

Fields

NAME	DESCRIPTION
DynamicSegment	The segment is part of the dynamic part
StaticSegment	The segment is part of the static part

Namespace FlexConfig.Sdk.Exceptions

Classes

FcBaseApiErrorException

Is thrown, when fcBase Api DLL code returns an error uint.

NotSupportedDeviceException

Is thrown, when a device with a not supported DeviceType was found.

Class FcBaseApiErrorException

Is thrown, when fcBase Api DLL code returns an error uint.

Inheritance

Object
Exception
FcBaseApiErrorException

Implements

ISerializable

Inherited Members

Exception.GetBaseException()
Exception.GetObjectData(SerializationInfo, StreamingContext)
Exception.GetType()
Exception.ToString()
Exception.Data
Exception.HelpLink
Exception.HResult
Exception.InnerException
Exception.Message
Exception.Source
Exception.StackTrace
Exception.TargetSite
Exception.SerializeObjectState
Object.Equals(Object)
Object.Equals(Object, Object)
Object.GetHashCode()
Object.MemberwiseClone()
Object.ReferenceEquals(Object, Object)

Namespace: FlexConfig.Sdk.Exceptions

Assembly: FlexConfig.Sdk.dll

Syntax

```
public class FcBaseApiErrorException : Exception, ISerializable
```

Constructors

FcBaseApiErrorException()

Initializes a new instance of the FcBaseApiErrorException class.

Declaration

```
public FcBaseApiErrorException()
```

FcBaseApiErrorException(String)

Initializes a new instance of the FcBaseApiErrorException class.

Declaration

```
public FcBaseApiErrorException(string message)
```

Parameters

TYPE	NAME	DESCRIPTION
String	message	Error code, text and type from fcBase Api.

FcBaseApiErrorException(String, Exception)

Initializes a new instance of the FcBaseApiErrorException class.

Declaration

```
public FcBaseApiErrorException(string message, Exception inner)
```

Parameters

TYPE	NAME	DESCRIPTION
String	message	Error code, text and type from fcBase Api.
Exception	inner	Caught Exception to be rethrown.

Implements

System.Runtime.Serialization.ISerializable

Class NotSupportedDeviceException

Is thrown, when a device with a not supported DeviceType was found.

Inheritance

Object
Exception
NotSupportedDeviceException

Implements

ISerializable

Inherited Members

Exception.GetBaseException()
Exception.GetObjectData(SerializationInfo, StreamingContext)
Exception.GetType()
Exception.ToString()
Exception.Data
Exception.HelpLink
Exception.HResult
Exception.InnerException
Exception.Message
Exception.Source
Exception.StackTrace
Exception.TargetSite
Exception.SerializeObjectState
Object.Equals(Object)
Object.Equals(Object, Object)
Object.GetHashCode()
Object.MemberwiseClone()
Object.ReferenceEquals(Object, Object)

Namespace: FlexConfig.Sdk.Exceptions

Assembly: FlexConfig.Sdk.dll

Syntax

```
public class NotSupportedDeviceException : Exception, ISerializable
```

Constructors

NotSupportedDeviceException()

Initializes a new instance of the NotSupportedDeviceException class.

Declaration

```
public NotSupportedDeviceException()
```

NotSupportedDeviceException(String)

Initializes a new instance of the NotSupportedDeviceException class.

Declaration

```
public NotSupportedDeviceException(string message)
```

Parameters

TYPE	NAME	DESCRIPTION
String	message	Exception message.

NotSupportedDeviceException(String, Exception)

Initializes a new instance of the NotSupportedDeviceException class.

Declaration

```
public NotSupportedDeviceException(string message, Exception inner)
```

Parameters

TYPE	NAME	DESCRIPTION
String	message	Exception message.
Exception	inner	Caught Exception to be rethrown.

Implements

System.Runtime.Serialization.ISerializable

Namespace FlexConfig.Sdk.Interpreter

Interfaces

IArrayDataInterpreter

Defines the interface to interpret the data type, which is defined as an array. This interface implements the IDataTypeInterpreter.

ICommonDataTypeInterpreter

Defines the interface to interpret the common data type information of a SOME/IP service. This interface implements the IDataTypeInterpreter.

IComplexDataTypeInterpreter

Defines the interface to interpret the complex data type information of a SOME/IP service. This interface implements the IDataTypeInterpreter.

IDataTypeInterpreter

Defines the interface to interpret SOME/IP service data type information. The data type interpreter is set inside a IServiceElement. Depending on the data type, the interpreter can be a ICommonDataTypeInterpreter, IComplexDataTypeInterpreter or a IArrayDataInterpreter.

IFrameInterpreter

Defines the interface to interpret frame information. The frame interpreter is set inside a IFrameEvent.

IPduInterpreter

Defines the interface for interpreting a IPdu.

ISignalInterpreter

Defines the interface to interprets a signal.

ISignalValue

Defines the interface used to keep the information of a interpreted signal.

Enums

DataTypeInterpreterType

Defines the data type interpreter type.

Enum DataTypeInterpreterType

Defines the data type interpreter type.

Namespace: FlexConfig.Sdk.Interpreter

Assembly: FlexConfig.Sdk.dll

Syntax

```
public enum DataTypeInterpreterType
```

Fields

NAME	DESCRIPTION
Array	The data type is defined as an array.
Common	The data type is a common data type.
Complex	The data type is a complex data type.

Interface IArrayDataInterpreter

Defines the interface to interpret the data type, which is defined as an array. This interface implements the IDataTypeInterpreter.

Inherited Members

IDataTypeInterpreter.Payload
IDataTypeInterpreter.DataType
IDataTypeInterpreter.ShortName
IDataTypeInterpreter.ArrayIndexes
IDataTypeInterpreter.InterpreterType

Namespace: FlexConfig.Sdk.Interpreter

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IArrayDataInterpreter : IDataTypeInterpreter
```

Properties

IndexedInterpreters

Gets the all indexed interpreters.

Declaration

```
IEnumerable<IDataTypeInterpreter> IndexedInterpreters { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<IDataTypeInterpreter>	The indexed interpreters.

Methods

GetIndexedInterpretedAt(Int32[])

Gets the interpreted data types with the given index value(s).

Declaration

```
IDataTypeInterpreter GetIndexedInterpretedAt(params int[] parameters)
```

Parameters

TYPE	NAME	DESCRIPTION
Int32[]	parameters	The parameters.

Returns

TYPE	DESCRIPTION
IDataTypeInterpreter	Th interpreter of the given index(es).

Interface ICommonDataTypeInterpreter

Defines the interface to interpret the common data type information of a SOME/IP service. This interface implements the IDataTypeInterpreter.

Inherited Members

IDataTypeInterpreter.Payload
IDataTypeInterpreter.DataType
IDataTypeInterpreter.ShortName
IDataTypeInterpreter.ArrayIndexes
IDataTypeInterpreter.InterpreterType

Namespace: FlexConfig.Sdk.Interpreter

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface ICommonDataTypeInterpreter : IDataTypeInterpreter
```

Methods

GetValue()

Gets the signal value object, which contains the raw and interpreted values.

Declaration

```
ISignalValue GetValue()
```

Returns

TYPE	DESCRIPTION
ISignalValue	The signal value object with raw and interpreted values.

Interface IComplexDataTypeInterpreter

Defines the interface to interpret the complex data type information of a SOME/IP service. This interface implements the IDataTypeInterpreter.

Inherited Members

IDataTypeInterpreter.Payload
IDataTypeInterpreter.DataType
IDataTypeInterpreter.ShortName
IDataTypeInterpreter.ArrayIndexes
IDataTypeInterpreter.InterpreterType

Namespace: FlexConfig.Sdk.Interpreter

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IComplexDataTypeInterpreter : IDataTypeInterpreter
```

Properties

Interpreters

Gets the interpreters of the sub data types.

Declaration

```
IEnumerable<IDataTypeInterpreter> Interpreters { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<IDataTypeInterpreter>	The interpreters.

Interface IDataTypeInterpreter

Defines the interface to interpret SOME/IP service data type information. The data type interpreter is set inside a IServiceElement. Depending on the data type, the interpreter can be a ICommonDataTypeInterpreter, IComplexDataTypeInterpreter or a IArrayDataInterpreter.

Namespace: FlexConfig.Sdk.Interpreter

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IDataTypeInterpreter
```

Properties

ArrayIndexes

Gets the array indexes. This field is relevant just by the sub items of arrays.

Declaration

```
IEnumerable<int> ArrayIndexes { get; }
```

Property Value

TYPE	DESCRIPTION
IEnumerable<Int32>	The array indexes.

DataType

Gets or sets the type of the data.

Declaration

```
IDataType DataType { get; set; }
```

Property Value

TYPE	DESCRIPTION
IDataType	The type of the data.

InterpreterType

Gets the type of the interpreter.

Declaration

```
DataTypeInterpreterType InterpreterType { get; }
```

Property Value

TYPE	DESCRIPTION
DataTypeInterpreterType	The type of the interpreter.

Payload

Gets the raw payload of the data type. Use the 'ToArray' method in order to get a byte array.

Declaration

```
ReadOnlySpan<byte> Payload { get; }
```

Property Value

TYPE	DESCRIPTION
ReadOnlySpan<Byte>	The raw payload.

ShortName

Gets the short name of the data type declaration.

Declaration

```
string ShortName { get; }
```

Property Value

TYPE	DESCRIPTION
String	The short name.

Interface IFrameInterpreter

Defines the interface to interpret frame information. The frame interpreter is set inside a IFrameEvent.

Namespace: FlexConfig.Sdk.Interpreter

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IFrameInterpreter
```

Methods

GetPdus()

Gets a list of interpreted Pdus, which the message actually contains.

Declaration

```
IEnumerable<IPduInterpreter> GetPdus()
```

Returns

TYPE	DESCRIPTION
IEnumerable<IPduInterpreter>	A list of interpreted Pdus.

Interface IPduInterpreter

Defines the interface for interpreting a IPdu.

Namespace: FlexConfig.Sdk.Interpreter

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface IPduInterpreter
```

Properties

Payload

Gets the raw payload of the PDU. Use the 'ToArray' method in order to get a byte array.

Declaration

```
ReadOnlySpan<byte> Payload { get; }
```

Property Value

TYPE	DESCRIPTION
ReadOnlySpan<Byte>	The raw payload.

Pdu

Gets the Gets the PDU object, which is defined in the database.

Declaration

```
IPdu Pdu { get; }
```

Property Value

TYPE	DESCRIPTION
IPdu	The database PDU.

UpdateBit

Gets the update bit value of the PDU. Consider following values: '1': The update bit is set. '0': The update bit is not set. '-1': The PDU have no update bit.

Declaration

```
int UpdateBit { get; }
```

Property Value

TYPE	DESCRIPTION
Int32	The value of the update bit or '-1' if the PDU has no update bit.

Methods

GetSignals()

Gets a list of interpreted signals, which the PDU actually contains.

Declaration

```
IEnumerable<ISignalInterpreter> GetSignals()
```

Returns

TYPE	DESCRIPTION
IEnumerable<ISignalInterpreter>	A list of interpreted signals.

Interface ISignalInterpreter

Defines the interface to interprets a signal.

Namespace: FlexConfig.Sdk.Interpreter

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface ISignalInterpreter
```

Properties

Signal

Gets the signal for which this interpreter is responsible.

Declaration

```
ISignal Signal { get; }
```

Property Value

TYPE	DESCRIPTION
ISignal	The signal.

Methods

GetValue()

Gets the signal value object, which contains the raw and interpreted values.

Declaration

```
ISignalValue GetValue()
```

Returns

TYPE	DESCRIPTION
ISignalValue	The signal value object with raw and interpreted values.

GetValueFromPhysical(Double)

Gets the signal value from a physical value.

Declaration

```
ISignalValue GetValueFromPhysical(double physicalValue)
```

Parameters

TYPE	NAME	DESCRIPTION
Double	physicalValue	The new physical value.

Returns

TYPE	DESCRIPTION
ISignalValue	The signal value object.

GetValueFromText(String)

Gets the signal value from a text value.

Declaration

```
ISignalValue GetValueFromText(string textValue)
```

Parameters

TYPE	NAME	DESCRIPTION
String	textValue	The new text value.

Returns

TYPE	DESCRIPTION
ISignalValue	The signal value object.

Interface ISignalValue

Defines the interface used to keep the information of a interpreted signal.

Namespace: FlexConfig.Sdk.Interpreter

Assembly: FlexConfig.Sdk.dll

Syntax

```
public interface ISignalValue
```

Properties

InternalValue

Gets the internal value of the signal.

Declaration

```
double InternalValue { get; }
```

Property Value

TYPE	DESCRIPTION
Double	The internal value.

InterpretedValue

Gets or sets the interpreted physical value of the signal.

Declaration

```
double InterpretedValue { get; set; }
```

Property Value

TYPE	DESCRIPTION
Double	The interpreted value.

IsInternalDefined

Gets a value indicating whether this instance has a internal value.

Declaration

```
bool IsInternalDefined { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	`true` if this instance has a internal value; otherwise, `false`.

IsPhysicalDefined

Gets a value indicating whether this instance has a physical value.

Declaration

```
bool IsPhysicalDefined { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	`true` if this instance has a physical value; otherwise, `false`.

IsRawDefined

Gets a value indicating whether this instance has a raw value.

Declaration

```
bool IsRawDefined { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	`true` if this instance has a raw payload; otherwise, `false`.

IsTextDefined

Gets a value indicating whether this instance has a text value.

Declaration

```
bool IsTextDefined { get; }
```

Property Value

TYPE	DESCRIPTION
Boolean	`true` if this instance has a text value; otherwise, `false`.

RawValue

Gets or sets the raw value of the signal. In case the signal raw value is larger than 8 bits, the byte order is increasing, e.g for a 12 Bit Raw value the following is used: Byte 0 contains Bit 0 .. 7 Byte 1 contains Bit 8 .. 11.

Declaration

```
IList<byte> RawValue { get; set; }
```

Property Value

TYPE	DESCRIPTION

TYPE	DESCRIPTION
IList<Byte>	The raw value.

TextValue

Gets or sets (if possible) the interpreted value as text.

Declaration

```
string TextValue { get; set; }
```

Property Value

TYPE	DESCRIPTION
String	The text value.

Unit

Gets the unit of the signal.

Declaration

```
string Unit { get; }
```

Property Value

TYPE	DESCRIPTION
String	The unit of the.

Release Notes

FL3X Config SDK 1.5.2.0

Release Date:
09 February 2023

Description: This is a minor release and compatible with the previous release.

It contains the following changes:

- Changed: Rename product name to FL3X Config SDK
- Changed: WIBU / Codemeter updated to 7.51
- Updated: Database converters
- Fixed: FL3X Device: Incorrect timestamp if measurement is started/stopped multiple (#428271)
- Fixed: FL3X Device: CAN HS Timestamping has failed sometimes in high bus load situation. Requires a RBS image version >= 5.5.10.1 (#431207)

FlexConfig SDK 1.4.3.0

Release Date:
03 June 2022

Description: This is a minor release and compatible with the previous release.

It contains the following changes:

- Added: Option for automatic distribution image update
- Added: improve database load times with caching option
- Updated: Database converters
- Changed: WIBU installation package updated to 7.40a
- Changed: Frames which has no triggering information are automatically removed from the IBus.Frames collection.
- Fixed: Receiving ethernet frames could crash application due to heap corruption (#420500)
- Fixed: FlexRay: the frame interpreter sets the correct frame also for channel B.(#414143)
- Fixed: FlexRay: On FlexDevice it may happened that an erroneous fifo configuration caused frame receiving problems (sometimes Channel A has the payload of Channel B)
- Fixed: Incorrect payload was used for interpreting big endian pdu segments.
- Fixed: add shortname/id information for frame interpreter contained pdus. (#420378)
- Fixed: Transmit overflow are now reported. (#381372)
- Fixed: Calculation of the "sign bit"-position was incorrect for signals greater than 32 bits. (#423138)
- Fixed: For Linear computation method the ICompuMethod.Scaling with only one element is not used to perform range checks for the internal value. (#423827)

FlexConfig SDK 1.3.2.0

Release Date:
20 December 2021

Description: This is a maintenance release and compatible with the previous release.

It contains the following changes:

- Fixed: Udp frames without vlan tag were not interpreted
- Fixed: VLAN tags handling improved
- Fixed: Improved Ethernet frame database handling (some frame has been missing)
- Fixed: Unsorted packets may causing incorrect timestamp overflow [409979]
- Fixed: Correct calculation of physical value if internal value is a floating type [408899]
- Fixed: DBC-Converter: Fix importing units [408776]
- Fixed: Loading database takes too long due to problems finding ethernet service frames [406381]

- Fixed: Signal is now correctly identified (avoids problems with duplicate signals in the same pdu).
- Fixed: Frame interpreter now handles frames without payload data correctly e.g FlexRay Nullframes [413192]
- Updated: Database converters

FlexConfig SDK 1.2.4.0

Release Date:
09 September 2021

Description: This is a maintenance release and compatible with the previous release.

It contains the following changes:

- Fixed: Missing Dlls in the delivery package [406558]

FlexConfig SDK 1.2.3.0

Release Date:
24 June 2021

Description: This is a minor release with new features and compatible with the previous release.

It contains the following changes:

- Added: Support for monitoring automotive ethernet channels (FlexDevice)
- Added: Support for SOME/IP interpretation
- Updated: Database converters

FlexConfig SDK 1.1.0.0

Release Date:
10 December 2020

Description: This is a minor release with new features and compatible with the previous release.

It contains the following changes:

- Added: Support for CAN / CAN FD Transmit
- Updated: Database converters

FlexConfig SDK 1.0.5.393

Release Date:
22 October 2020

Description: This is a maintenance release and compatible with the previous release.

It contains the following changes:

- Fixed: High cpu utilization
- Fixed: High memory utilization [366177]
- Fixed: Physical constrains hasn't been parsed correctly. [378207]
- Fixed: Signal interpretation was not functional in some cases. [378933]
- Changed: Additional AppDomain fixes from WIBU [367109]
- Changed: WIBU installation package updated to 7.10a due to security vulnerabilities. See https://www.wibu.com/de/support/security-advisories.html for details.

FlexConfig SDK 1.0.5.0

Release Date:
13 May 2020

Description: This is a maintenance release and compatible with the previous release.

It contains the following changes:

- Changed: Minor corrections in the documentation
- Changed: AppDomain support for WIBU protection enabled

FlexConfig SDK 1.0.4.0

Release Date:
06 April 2020

Description: This is the first official release of the FlexConfig Sdk.

It contains the following changes:

- Added database format support for FIBEX+ 1.4, FIBEX 2.0.0d, FIBEX 2.0.1, FIBEX 3.0.0, FIBEX 3.1.0, FIBEX 4.0.0, FIBEX 4.1.0, FIBEX 4.1.1, AUTOSAR 3.0.2, AUTOSAR 3.1.0, AUTOSAR 3.1.4.DAI.2, AUTOSAR 3.1.4.DAI.4, AUTOSAR 3.2.2, AUTOSAR 4.0.3, AUTOSAR 4.1.1, AUTOSAR 4.2.1, AUTOSAR 4.2.2, AUTOSAR 4.3.0 and DBC.
- Added device support for FlexDevice-S, FlexDevice L, FlexDevice L^2, FlexCard PMC-II and FlexCard USB-M
- Added device enumeration functionality
- Added simple device configuration via database file
- Added bus support for CAN, CAN-FD and FlexRay
- Added bus frame receive functionality
- Added bus frame interpretation functionality for PDU and Signals.
- Added support for PDU Container and Multiplexed PDU.

Version compatibility

FL3X CONFIG SDK	FCBASE	PC-HW-IF DLL	PC-HW-IF EMBEDDED (ANALYZER)	.NET
1.5.2.0	6.9+	3.0+	3.0+ (1.2+)	.NET Standard 2.0
1.4.3.0	6.8.13.0	3.0+	3.0+ (1.2+)	.NET Standard 2.0
1.3.2.0	6.8.3.0	2.5	2.5 (1.2+)	.NET Standard 2.0
1.2.4.0	6.7.5.0	2.5	2.5 (1.2+)	.NET Standard 2.0
1.2.3.0	6.7.5.0	2.5	2.5 (1.2+)	.NET Standard 2.0
1.1.0.0	6.7.0.0	2.4	2.4 (1.1.0.1)	.NET Standard 2.0
1.0.5.0	6.6.6.0	2.3.2.2 (Rev 1973 HwCom)	2.3.1.0 (1.0.0.21)	.NET Standard 2.0
1.0.4.0	6.6.6.0	2.3.1.2 (Rev 1953 HwCom)	2.3.1.0 (1.0.0.21)	.NET Standard 2.0

Open issues

FL3X Config SDK 1.5.2.0

Release Date:
09 February 2023

Dear Ladies and Gentlemen,

Thank you for your interest in the software FL3X Config SDK. To ensure your customer satisfaction we would like to point out a few functional limitations, which will be corrected with timely product updates.

For questions please contact our support team directly.

Your STAR ELECTRONICS Sales Team

General Open Issues:

NUMBER	COMPONENT	RESTRICTIONS	EFFECTS
366176	FL3X Device-S/L/L²	Frame loss	In high bus load situations it may happen that bus frames are lost due to internal/operation system buffer are overflowing. Currently this loss is not indicated but in future updates there will be a indication for a frame loss.
381732	FL3X Device-S/L/L²	Bus communication not working	Incorrect tiny configuration are not detected properly which results that FL3X Config SDK can not establish a valid bus communication.
402812	FL3X Device-S/L/L²	fcbEthSetFilterConfiguration not functional	ETH set filter function is not functional, ETH filter can not be set in use with FL3X Device hardware (FlexMeasure)

License

This chapter contains all license agreements for FL3X Config SDK. / Dieses Kapitel beinhaltet alle Lizenzvereinbarungen für FL3X Config SDK.

Published by / Herausgeber: STAR ELECTRONICS GmbH & Co. KG

Contents / Inhalt

1. Terms and Conditions for Software Use in English (11 November 2015)

2. Softwarenutzungsbedingungen in Deutsch (11. November 2015)

3. Third Party Licenses / Lizenzvereinbarungen von Dritten

License of the used Serilog Library / Lizenz der verwendeten Serilog Bibliothek

(I) Apache License Version 2.0

License of the used NETStandard Library / Lizenz der verwendeten NETStandard Bibliothek

(II) MIT License

License of the used PacketDotNet Library / Lizenz der verwendeten PacketDotNet Bibliothek

(III) Mozilla Public License Version 2.0

Terms and Conditions for Software Use

§ 1 Scope

1. These Terms and Conditions for Software Use shall govern STARs provision of Software to the Customer.
2. If the Customer acquires Hardware from STAR in which the Software has been preinstalled, these Terms and Conditions for Software Use shall apply to the preinstalled copy of the Software by way of analogy.
3. These Terms and Conditions for Software Use govern the provision of software against payment and the provision of Demo Software. However, the provision of Demo Software is only governed to the extent explicitly provided in § 20 of these Terms and Conditions for Software Use. Provisions of these Terms and Conditions for Software Use that § 20 does not explicitly declare to be applicable to the provision of Demo Software apply exclusively to software that is provided against payment.

§ 2 Definitions

Unless otherwise explicitly stipulated, the following terms used in these Terms and Conditions for Software Use (including the definitions themselves) have the following meaning:

Customer

The party to the respective agreement with STAR as licensee.

Demo Software

Software that STAR provides to the Customer free of charge which has a limited functional scope compared to the corresponding Software provided against payment.

Device Driver

Software that is designed to control a specific piece of Hardware and is acquired by the Customer from STAR together with this Hardware. Whether or not a piece of Software is a Device Driver is set forth in the respective User Documentation.

Download

The possibility of transferring data in the Internet from one computer to the Customer's computer.

STAR

STAR ELECTRONICS GmbH & Co. KG.
Jahnstraße 86
73037 Goeppingen (Germany)
Register Court: Ulm, HRA-Nr. 721096
www.flex-product.com

Firmware/Embedded Software

Software that is embedded in Hardware and is acquired by the Customer from STAR together with this Hardware. Whether such Software is Firmware or Embedded Software is set forth in the respective User Documentation.

Hardware

Electromechanical devices that are generally not functional without Software.

License Materials

The Software together with the User Documentation.

License Models

The License Models described in § 6 of these Terms and Conditions for Software Use for the licensing of the Software.

Normal Working Hours

Monday through Thursday 9:00 a.m. – 12:00 noon and 1:00 p.m. – 3:30 p.m. plus Friday 09:00 a.m. – 12:00 noon Central European Time (daylight-saving time), excepting legal holidays in the Federal States of Baden-Württemberg or Bavaria and STAR's general closing times (in particular 24 December through 6 January).

Object Code

The Software in a form that is only machine readable which is suitable for generating an executable machine code.

Parties

STAR and the Customer together.

Software

The software that is listed and described in the order confirmation.

Source Code

The source code of the Software that is readable by humans and written in a programming language.

Terms and Conditions for Software Use

These general terms and conditions for the provision of Software for use.

Update

The bundling of several defect corrections and/or repairs of faults in the Software, as well as minor functional improvements or adjustments of the Software in a single batch.

User Documentation

The textual and technical description of the Software that is in effect prior to the conclusion of this agreement and is provided to the Customer prior to the conclusion of this agreement (e.g. Instructions for Use, User Manual, Documentation).

§ 3 Order of priority

1. The agreements entered into by the Customer and STAR have the following order of priority:

   – order confirmation

   – User Documentation on the respective Software

   – these Terms and Conditions for Software Use

   – STAR's general terms and conditions of sale

   In the event of any conflicts, the provisions listed first will always take priority over those listed last. Any gaps shall be filled by the respective subordinate provisions.

2. Any contrary or differing terms and conditions of the Customer shall only apply with STAR's explicit written consent.

§ 4 Contract formation

All offers by STAR are non-binding. A contract between the Customer and STAR will only be concluded by the written confirmation of an order or the delivery of Software by STAR.

§ 5 General Terms and Conditions of use rights

1. STAR grants the Customer the right to use the Software under these Terms and Conditions for Software Use. For the use of the Software, STAR offers the License Models designated in § 6 of these Terms and Conditions for Software Use. The following provisions of this § 5 of the Terms and Conditions for Software Use shall apply to all License Models insofar as no other provision is expressly made under § 6 of these Terms and Conditions for Software Use. Any use or exploitation of the Software that goes beyond these Terms and Conditions for Software Use is prohibited.

2. For the duration of the existence of any intellectual property rights, the Customer is granted a non-exclusive right to use the Software for internal use for the purposes set forth in the User Documentation. Use of Software means any permanent or temporary full or partial reproduction of the Software by means of saving, loading, running or displaying the Software for the purpose of executing the Software; any other reproduction of the Software is prohibited, with the exception of the making of a backup copy (§ 7 of these Terms and Conditions for Software Use). Prohibited is in particular, but without limitation, the reproduction of the Software for the purpose of passing it on to third parties with or without remuneration.

3. The Customer may only modify or edit the Software to the extent necessary for the use described in the User Documentation, to connect the Software with another computer program, or to correct errors. Other than that, the Customer may only modify the Software if and to the extent STAR has explicitly permitted the Customer to do so in writing.

4. With regard to the rights to the Software granted to the Customer, the Customer shall not grant rights to third parties. Nor may the Customer grant third parties a sublicense in the rights to the Software granted to the Customer. § 18 of these Terms and Conditions for Software Use remains unaffected.

§ 6 License Models

1. STAR offers the Customer the following License Models for the licensing of the Software:

   a. "Single User License" Only one employee of the Customer may use the Software on one computer at a time. The Customer is free to choose which computer and which employee will be involved in the use. The Customer is not allowed to keep the Software on the server of a network if this would enable multiple uses of the Software at the same time. If the Customer wishes to have the Software used by multiple employees or on multiple computers at the same time, the Customer must notify this in writing in advance and purchase additional licenses for the Software from STAR.

   b. "Floating License" The Customer may have the Software used by two or more employees on various computers at the same time. The number of employees entitled to use it at the same time can be derived from the order confirmation. The Customer is free to choose which employees and which computers will be involved in the use. If the Customer wishes to

increase the number of employees entitled to use the Software at the same time, it must notify this in writing in advance and purchase additional licenses for the Software from STAR.

c. "Integrated Software License" The Integrated Software License applies to device drivers, Firmware and Embedded Software. The Customer may use the licensed Software under the Integrated Software License only for the designated operation of the Hardware acquired from STAR together with this Software. The designated operation The designated operation is set forth in the respective User Documentation. Any other use of the Software, in particular but not limited to, the use of the Software for Hardware that the Customer did not acquire from STAR together with the Software is prohibited.

d. "Development Kit License" The Customer may use the Software for the project stated in the order confirmation. Should the Customer wish to use the Software for other projects, the Customer must make prior notification of this to STAR and purchase further licenses for the software. Should no project be stated in the order confirmation in spite of the license model "Development Kit License" being specified, the Development Kit License shall be valid for the Customer's project which comes closest to the subject matter of the license in terms of content. In the framework of the respective project, the Software licensed under the Development Kit License may,

- (i) in derogation of § 5 para. 2 of these Terms and Conditions for Software Use, be used not only for internal purposes but also for third-party purposes.
- (ii) in derogation of § 5 para. 2 of these Terms and Conditions for Software Use, be reproduced for the purpose of passing on – either with or without remuneration – software developed by the Customer. In this respect, passing on Software to third parties is subject to the condition that the Customer passes on the file "Copyright.txt" to the third party, and that the third party undertakes to comply with the restrictions deriving from the order confirmation, the User Documentation, the Terms and Conditions for Software Use and the restrictions deriving from the file "Copyright.txt" in using the Software.
- (iii) in derogation of § 5 para. 4, be sublicensed to third parties insofar as the respective third party undertakes to comply with the restrictions deriving from the order confirmation, the User Documentation, these Terms and Conditions for Software Use and the restrictions deriving from the file "Copyright.txt" in using the Software. Any sublicense is dependent on the main license as regards validity and scope.

2. Device drivers, Firmware and Embedded Software shall always be licensed under the Integrated Software License. Otherwise, the rule shall apply that unless otherwise explicitly regulated in the confirmation of order, the Customer will merely be acquiring a "Single User License".

§ 7 Backup copy

The use of the Software and the data contained therein in accordance with these Terms and Conditions for Software Use includes the creation of a backup copy. The backup copy may only be used for backup purposes. The Customer shall keep the backup copy at a secure location that is not accessible to third parties.

§ 8 Source Code/Object Code

1. The Software will be provided to the Customer in the Object Code as a Download or on machine readable data carriers.
2. Unless otherwise explicitly provided in the order confirmation, the Customer is not entitled to the provision of the Source Code or parts thereof.
3. A decompilation of the Object Code is permissible only in compliance with the statutory limitations set forth in sec. 69e of the German Copyright Act (UrhG). Any decompilations in excess thereof are prohibited.

§ 9 User Documentation

1. The Customer will receive one User Documentation in typewritten form and/or in electronic form on machine readable data carriers or as a Download.
2. The Customer may reproduce the User Documentation in typewritten form only with STAR's prior written consent. If the

Customer receives the User Documentation only in electronic form, then notwithstanding sentence 1 the Customer may make one copy of the User Documentation in typewritten form for its own internal use.

3. In all other respects, § 5 through § 7 of these Terms and Conditions for Software Use shall apply by analogy to the use of the User Documentation in electronic form.

§ 10 Protection of the License Materials

1. The Customer undertakes to comply with the protection notices contained in the License Materials, such as copyright notices and other reservations of rights, to retain them in unmodified form as well as to incorporate them into the backup copy in unmodified form.

2. Notwithstanding the use rights granted on the basis of these Terms and Conditions for Software Use, STAR retains all rights to the License Materials, including all rights in all copies or partial copies created thereof by the Customer. This does not affect the Customer's ownership of machine readable data carriers and Hardware.

§ 11 Maintenance of Software

1. A prerequisite for the provision of Software maintenance is the conclusion of a maintenance agreement for a consideration. Without a maintenance agreement, the Customer shall have no claim to Software maintenance.

2. Software maintenance comprises the delivery of Updates to the Software once STAR also provides it to other customers in the normal course of business.

3. Maintenance agreements shall have a basic term of one calendar year commenceing with the day on which the respective Software is provided. If the maintenance agreement is not terminated by one of the Parties in writing at least two calendar months prior to the expiration of the basic term, the maintenance agreement shall extend by another calendar year. Sentence 2 applies for every subsequent extension mutatis mutandis.

4. For the use of Updates of the Software, these Terms and Conditions for Software Use, in the version current at the time of the delivery of the respective Updates shall apply.

§ 12 Warranty of title (Rechtsmängelgewährleistung)

1. The Customer shall notify STAR in writing without delay of any claims asserted against the Customer arising from an infringement of intellectual property rights by the License Materials.

2. If the Customer's designated use of the License Materials is impaired by the intellectual property rights of third parties, STAR pächer shall to an extent reasonable for the Customer, have the right at STAR's option to acquire corresponding licenses from third parties at STAR's expense or to modify or replace the License Materials such that they no longer impair the intellectual property rights of third parties.

3. STAR shall not be liable under a warranty of title where the infringement of intellectual property rights of third parties is due to the fact that the Customer has modified or edited the License Materials or is using the Software together with Hardware or computer programs that were not supplied by STAR. This also applies where the third party's claims arise from a use of the License Materials by the Customer after it had been informed by STAR that as a result of a third party claim the use should be stopped.

§ 13 Warranty of merchantability (Sachmängelgewährleistung)

1. The designated use and the conditions for operation of the Software are set forth in the User Documentation. The Customer bears sole responsibility for the selection and usage of the Software, including the performance results to be achieved by the usage of the Software.

2. STAR warrants that the Software in the version provided to the Customer is suitable for the designated use in accordance with the User Documentation. In the event of significant deviations from the User Documentation, STAR shall have the right and, unless unreasonable expense would be involved, the obligation to perform remediation. If the Customer cannot reasonably be expected to await the next Update of the Software and STAR fails to remediate such significant deviations of

the Software from the User Documentation or to work around them such that the Customer is able to use the Software in the designated manner within a reasonable period of time,, the Customer may, at its option, withdraw or instead claim a reduction of the license fee pursuant to sec. 441 of the German Civil Code (BGB).

3. STAR will fulfill its warranty obligations during the Normal Working Hours ; no costs will be incurred by the Customer in this respect. If STAR is to fulfill warranty obligations outside of normal working hours, the Parties shall enter into a written supplementary agreement to compensate for the additional costs.

4. The Customer is obligated to provide STAR with verifiable documentation of the nature and occurrence of deviations of the Software from the User Documentation and shall cooperate in isolating any defects.

5. Obvious defects must be notified to STAR within a period of 2 (two) weeks of receipt of the Software; otherwise, STAR shall not be liable for a warranty violation. The timely dispatch of the notification of defect will be sufficient for compliance with the deadline; the burden of proof is on the Customer. If the Customer is a merchant (Kaufmann), the provisions of sec. 377 of the German Commercial Code (HGB) shall apply.

6. STAR shall have no warranty obligations if the Customer has modified or edited the Software.

7. STAR will likewise have no warranty obligations for defects in quality if the defect in the Software is due to the fact that the Customer is using the Software together with hardware or computer programs that were not supplied by STAR.

8. Should STAR nonetheless perform work to remediate defects in the cases set forth in § 13 paras. 6 and 7 of these Terms and Conditions for Software Use, the Customer shall pay for such work at STAR's normal rates in effect at the time such work is carried out.

9. The above limitations of the warranty obligation shall not apply in cases in which STAR has assumed a guarantee for a condition of the License Material or if STAR fraudulently concealed any defects.

§ 14 Statute of limitations of warranty claims

Subject to the provisions in the following sentence and in § 13 para. 9 of these Terms and Conditions for Software Use, warranty claims will become time-barred one year after delivery of the Software. However, the Customer's claims for damages or reimbursement of expenditures shall not be affected by the above provision and shall become time-barred according to the statutory time periods; for claims for damages and reimbursement of expenditures, the provisions in § 15 of these Terms and Conditions for Software Use shall apply.

§ 15 Disclaimer of Liability

1. In the event of a breach of material contractual duties (so-called "Cardinal Obligations"), STAR shall be liable toward the Customer for reimbursement of expenditures and damages (hereinafter in this § 15 of the Terms and Conditions for Software Use collectively referred to as **"Damages"**). Cardinal Obligations are all obligations whose breach would jeopardize the achievement of the contractual purpose, as well as all obligations whose fulfillment enables the proper performance of the contract in the first place and on compliance with which the Customer can generally rely. However, where a breach of a Cardinal Obligation is due only to slight negligence and does not result in injury to life, limb or health, the Customer's claims will be limited to Damages in the amount of the typically foreseeable damage.

2. STAR shall also be liable to the Customer pursuant to the provisions of the German Product Liability Act (Produkthaftungsgesetz), in cases of intent and gross negligence, for injury to life, limb or health, if it assumes a guarantee, as well as in all other cases of mandatory statutory liability, in each case in accordance with the statutory provisions.

3. In all other cases, claims for damages against STAR – regardless of their legal basis, in particular due to a breach of obligations under the contractual relationship between the Parties by STAR, its legal representatives, employees or vicarious agents (Erfüllungsgehilfen), pursuant to sec. 311a German Civil Code or based on tort – are disclaimed.

4. To the extent STAR's liability is limited or disclaimed under the above provisions, this shall also apply to the personal liability of STAR's legal representatives, employees and vicarious agents.

§ 16 Operating conditions for the Software

1. The Software provided to the Customer was developed for use on certain hardware and for purposes of interoperability with certain other computer programs. These operating conditions are set forth in detail in the respective User Documentation, compliance with which is mandatory for the Customer. The Customer shall also inspect the Software carefully before putting it into productive operation. This applies especially, but not only, if the Software is to be put to use in a safety-critical area.

2. If the Software is not used in compliance with the operating conditions as set forth in § 16 para. 1 of these Terms and Conditions for Software Use, the warranty obligation pursuant to § 12 and § 13 of these Terms and Conditions for Software Use shall not apply. Should STAR nonetheless perform work to remediate the defects, the Customer shall pay for such work at STAR's normal rates in effect at the time such work is carried out.

3. In the event that third parties assert claims against STAR arising from the fact that the Customer, its legal representatives, employees or vicarious agents culpably used the Software without complying with the operating conditions pursuant to § 16 para. 1 of these Terms and Conditions for Software Use, the Customer shall indemnify STAR against such claims of the third party and reimburse STAR for all damage incurred in this connection. STAR shall notify the Customer in writing without delay in case any claims within the meaning of sentence 1 are asserted against it.

§ 17 Test phase

1. The Customer has the right to test the Software free of charge before putting it into productive operation if this is explicitly provided for in the order confirmation. In that case, the individual modalities of the test phase (duration, etc.) will be those stated in the order confirmation.

2. In any case, the test phase shall end once the Customer begins to put the Software into productive operation. The Customer shall notify STAR in writing of the commencement of the productive operation.

§ 18 Transfer of the License Materials

1. The Customer has the right to transfer the License Materials in their original condition to a third party, together with a copy of these Terms and Conditions for Software Use and the order confirmation as long as the Customer provides STAR with the name and complete address of such third party and the third party undertakes to STAR to comply with the restrictions set forth in the order confirmation, the User Documentation and these Terms and Conditions for Software Use when using the Software. This right to transfer the License Materials does not extend to modified or edited versions of the License Materials.

2. Upon the transfer of the License Materials in compliance with the prerequisites set forth in § 18 para. 1 of the Terms and Conditions for Software Use, the right to use the License Materials will pass to the third party, who thereby replaces the Customer within the meaning of these Terms and Conditions for Software Use. At the same time, the Customer's right to use the Software will end.

3. Upon the transfer of the License Materials, the Customer shall immediately and completely delete or otherwise destroy all copies and partial copies of the License Materials as well as modified or edited versions of the License Materials. This also applies to the backup copy pursuant to § 7 of these Terms and Conditions for Software Use.

4. § 18 paras. 1 to 3 of these Terms and Conditions for Software Use also apply if the transfer consists of a temporary provision for use. However, it is prohibited to rent out the License Materials or parts thereof.

5. For cases in which the License Materials are subsequently transferred to other third parties, § 18 paras. 1 to 4 of these Terms and Conditions for Software Use shall apply accordingly.

6. The above provisions of this § 18 of the Terms and Conditions for Software Use do not apply to Software licensed under the License Model "Development Kit License". Passing on this Software to third parties is only permissible subject to the preconditions provided for under § 6 para. 1 lit. d of the Terms and Conditions for Software Use.

§ 19 License fee / maintenance fees

1. The Customer shall pay the license and maintenance fees set forth in the order confirmation. The amounts listed in the order confirmation do not include VAT, which will be invoiced separately at the rate in effect at the time.

2. In all other respects, the payment of the license and maintenance fees will be governed by STAR's general terms and conditions of sale, which shall be provided to the Customer prior to the conclusion of the contract.

§ 20 Demo Software

1. Besides the Software STAR offers against payment, STAR also offers Demo Software in Object Code on machine readable data carriers or as a Download. The Customer is not entitled to a provision of Demo Software for use, nor to a provision of the Source Code of the Demo Software or parts thereof for use, nor to a provision of Updates of Demo Software for use.

2. The contract between the Customer and STAR on the provision of Demo Software for use shall be concluded by the utilization of the Demo Software by the Customer.

3. For the duration of the existence of any intellectual property rights, the Customer is granted a non-exclusive right to use the Demo Software internally for the purposes set forth in the User Documentation. Use of the Demo Software means any permanent or temporary full or partial reproduction of the Demo Software by means of saving, loading, running or displaying it for the purpose of executing the Demo Software; any other reproduction of the Demo Software is prohibited, with the exception of the making of a backup copy (§ 7 of these Terms and Conditions for Software Use). Prohibited is in particular, but without limitation, the reproduction of the Demo Software for the purpose of passing it on to third parties with or without remuneration.

4. The Customer shall have no right to modify or edit the Demo Software. A decompilation of the Object Code is permissible only subject to the statutory restrictions set forth in sec. 69e Copyright Act. Any decompilations in excess thereof are prohibited.

5. With regard to the rights to the Demo Software granted to the Customer, the Customer shall not grant any rights to third parties. Nor may the Customer grant third parties a sub-license in the rights to the Demo Software granted to the Customer.

6. STAR shall only be liable for defects of title or quality of the Demo Software and the respective User Documentation if STAR has fraudulently concealed such defect. Apart from warranty obligations, STAR shall only be liable for intent and gross negligence. Any other liability of STAR is disclaimed. To the extent STAR liability is limited or disclaimed pursuant to the above provisions, this shall also apply to STAR's legal representatives, employees and vicarious agents.

7. The Demo Software provided to the Customer was developed for use on certain hardware and for purposes of interoperability with certain other computer programs. These operating conditions are set forth in detail in the respective User Documentation. In the event that third parties assert claims against STAR arising from the fact that the Customer, its legal representatives, employees or vicarious agents culpably used the Demo Software without complying with the operating conditions pursuant to § 20 para. 7 sentence 2 of these Terms and Conditions for Software Use, the Customer shall indemnify STAR against such claims of the third party and reimburse STAR for all damage incurred in this connection. STAR shall notify the Customer in writing without delay in case any such claims are asserted against it.

8. In all other respects, the following provisions of these Terms and Conditions for Software Use shall apply by analogy to the provision of the Demo Software: § 7, § 9 (with the modification that § 9 para. 3 shall refer to this § 20), § 10 and § 22.

§ 21 Confidentiality

1. The confidentiality obligations of both Parties follow from a separate confidentiality agreement. If the Parties have not entered into such a separate confidentiality agreement, the following paras. 2 to 4 shall apply.

2. The Parties undertake to maintain secrecy for an unlimited period of time with regard to all confidential information that becomes accessible to them in connection with this agreement. Confidential information is information that a Party has either marked or in any other way designated in writing as protected or confidential, or information that, in light of the circumstances of its disclosure, the recipient Party must reasonably identify as confidential. Confidential information includes in particular, but without limitation, the License Materials.

3. The above confidentiality obligations will not apply if and to the extent the respective item of information demonstrably (i) is generally known or has become generally known without a party's fault and without violation of these obligations of confidentiality, (ii) is or becomes state of the art, (iii) was already known to the recipient Party at the time of its disclosure, which must be proved by documents evidencing such knowledge, (iv) was or is lawfully made known or accessible to the recipient Party by a third party, (iv) has to be disclosed due to statutory provisions or enforceable official orders or court decisions. The burden of proving the existence of such an exception is borne by the respective recipient of the information. In any case, the respective affected Party must, to the extent possible, be informed in a timely manner prior to the disclosure of the information to third parties.

4. Both Parties will take suitable precautions to protect the confidential information of the respective other Party. Both Parties will disclose confidential information of the respective other Party to governing bodies, employees, consultants or sub-contractors only subject to this confidentiality obligation, which must likewise be imposed on the recipient.

§ 22 Miscellaneous

1. Should any individual provisions of these Terms and Conditions for Software Use be or become invalid, this shall not affect the validity of the other provisions.

2. For the purpose of handling the provision of the Software, STAR will store personal data of the Customer in an IT system and automatically process it. The data will not be used for any purposes in excess thereof. It will not be passed on to third parties.

3. The governing laws are the laws of the Federal Republic of Germany. The UN Convention on Contracts for the International Sale of Goods does not apply.

4. If the Customer is a merchant (Kaufmann), a legal entity under public law (juristische Person des öffentlichen Rechts) or a special fund under public law (öffentlich-rechtliches Sondervermögen), the exclusive legal venue for all disputes shall be Stuttgart. STAR is entitled, at its option, to file suit against the Customer at the latter's general legal venue.

Softwarenutzungsbedingungen

§ 1 Geltungsbereich

1. Die Softwarenutzungsbedingungen gelten für die Überlassung von Software durch STAR an den Kunden.

2. Erwirbt der Kunde von STAR Hardware, in der die Software vorinstalliert ist, gelten die Softwarenutzungsbedingungen auch für die vorinstallierte Kopie der Software.

3. Die Softwarenutzungsbedingungen regeln die entgeltliche Softwareüberlassung und die Überlassung von Demo Software, die Überlassung von Demo Software jedoch nur, soweit dies ausdrücklich in § 20 der Softwarenutzungsbedingungen vorgesehen ist. Bestimmungen der Softwarenutzungsbedingungen, die § 20 der Softwarenutzungsbedingungen nicht ausdrücklich auf die Überlassung von Demo Software für anwendbar erklärt, gelten ausschließlich für die entgeltliche Softwareüberlassung.

§ 2 Definitionen

Soweit nicht ausdrücklich anders geregelt, haben in den Softwarenutzungsbedingungen (einschließlich der Definitionen selbst) die nachfolgenden Begriffe jeweils die folgenden Bedeutungen:

Anwenderdokumentation

Die vor Vertragsschluss gültige und dem Kunden vor Vertragsschluss zur Verfügung stehende inhaltliche und technische Beschreibung der Software (z.B. Instruction for Use, User Manual, Documentation).

Demosoftware

Eine Software mit im Verhältnis zur entsprechenden entgeltlichen Software eingeschränktem Funktionsumfang, die STAR den Kunden unentgeltlich zur Verfügung stellt.

Download

Die Möglichkeit zur Übertragung von Daten im Internet von einem Computer zum Computer des Kunden.

STAR

STAR ELECTRONICS GmbH & Co. KG.
Jahnstraße 86
73037 Goeppingen (Deutschland)
Registergericht: Ulm, HRA-Nr. 721096
www.flex-product.com

Firmware/ Embedded Software

Eine Software, die in Hardware eingebettet ist und vom Kunden gemeinsam mit dieser Hardware von STAR erworben wird. Ob es sich bei einer Software um eine Firmware oder Embedded Software handelt, ergibt sich aus der jeweiligen Anwenderdokumentation.

Gerätetreiber

Eine Software, die zur Steuerung einer Hardware bestimmt ist und vom Kunden gemeinsam mit dieser Hardware von STAR erworben wird. Ob es sich bei einer Software um einen Gerätetreiber handelt, ergibt sich aus der jeweiligen Anwenderdokumentation.

Hardware

Elektromechanische Geräte, die in der Regel ohne Software nicht funktionstüchtig sind.

Kunde

Der jeweilige Vertragspartner von STAR als Lizenznehmer.

Lizenzmaterial

Die Software nebst Anwenderdokumentation.

Lizenzmodelle

Die in § 6 dieser Softwarenutzungsbedingungen für die Lizenzierung der Software genannten Lizenzmodelle.

Softwarenutzungsbedingungen

Diese allgemeinen Bedingungen für die Überlassung von Software.

Objektcode

Die Software in einer ausschließlich für Maschinen lesbaren Form, die für die Erzeugung eines ausführbaren Maschinencodes geeignet ist.

Quellcode

Der für Menschen lesbare, in einer Programmiersprache geschriebene Quell-Text der Software.

Software

Die in der Auftragsbestätigung einzeln aufgeführte und beschriebene Software.

Übliche Arbeitszeiten

Montags bis Donnerstags 09.00 – 12.00 Uhr und 13.00 – 15.30 Uhr sowie Freitags 09.00 – 12.00 Uhr Mitteleuropäischer (Sommer-)Zeit, ausgenommen gesetzliche Feiertage in den Bundesländern Baden-Württemberg oder Bayern und allgemeine Schließzeiten von STAR (insbesondere vom 24. Dezember bis zum 6. Januar jeweils einschließlich).

Update

Die Bündelung mehrerer Mängelbehebungen und/oder Störungsbeseitigungen der Software sowie geringfügige funktionale Verbesserungen oder Anpassungen der Software in einer einzigen Lieferung.

Vertragsparteien

STAR und der Kunde gemeinsam.

§ 3 Rangfolge

1. Die zwischen dem Kunden und STAR geschlossenen Vereinbarungen stehen in der folgenden Rangfolge:

   – Auftragsbestätigung

   – Anwenderdokumentation zur jeweiligen Software

   – Softwarenutzungsbedingungen

   – Allgemeine Verkaufsbedingungen von STAR

Die zuerst genannten Bestimmungen haben bei Widersprüchen stets Vorrang vor den zuletzt genannten Bestimmungen. Lücken werden durch die jeweils nachrangigen Bestimmungen ausgefüllt.

2. Entgegenstehende oder abweichende Geschäftsbedingungen des Kunden gelten nur, wenn STAR diesen ausdrücklich schriftlich zugestimmt hat.

§ 4 Vertragsschluss

Alle Angebote von STAR sind freibleibend. Ein Vertrag kommt zwischen dem Kunden und STAR erst durch die schriftliche Auftragsbestätigung oder durch die Lieferung der Software durch STAR zustande.

§ 5 Allgemeine Bedingungen zum Nutzungsrecht

1. STAR räumt dem Kunden das Recht ein, die Software unter diesen Softwarenutzungsbedingungen zu nutzen. Für die Softwarenutzung bietet STAR die in § 6 dieser Softwarenutzungsbedingungen genannten Lizenzmodelle an. Die nachfolgenden Regelungen dieses § 5 der Softwarenutzungsbedingungen gelten für alle Lizenzmodelle, soweit in § 6 dieser Softwarenutzungsbedingungen nicht ausdrücklich etwas anderes geregelt ist. Jede über diese Softwarenutzungsbedingungen hinausgehende Nutzung oder Verwertung der Software ist ausgeschlossen.

2. Der Kunde erhält für die Dauer des Bestehens etwaiger Schutzrechte ein nicht ausschließliches Recht, die Software für die in der Anwenderdokumentation vorgesehenen Zwecke für den intern Gebrauch zu nutzen. Nutzung der Software ist dabei jede dauerhafte oder vorübergehende ganze oder teilweise Vervielfältigung der Software durch Speichern, Laden, Ablaufen oder Anzeigen zum Zweck der Ausführung der Software; jede anderweitige Vervielfältigung der Software ist mit Ausnahme der Anfertigung einer Sicherungskopie (§ 7 der Softwarenutzungsbedingungen) ausgeschlossen. Ausgeschlossen ist insbesondere, aber nicht nur, die Vervielfältigung der Software zum Zweck der unentgeltlichen oder entgeltlichen Weitergabe an Dritte.

3. Der Kunde darf die Software nur ändern oder bearbeiten, soweit dies zu der in der Anwenderdokumentation vorgesehenen Nutzung, zur Verbindung der Software mit einem anderen Computerprogramm oder zur Fehlerkorrektur erforderlich ist. Darüber hinaus darf der Kunde die Software nur ändern, wenn und soweit STAR dies dem Kunden ausdrücklich schriftlich erlaubt hat.

4. Der Kunde darf hinsichtlich der ihm an der Software eingeräumten Rechte Dritten keine Rechte einräumen. Ebenfalls darf der Kunde die ihm an der Software eingeräumten Rechte nicht an Dritte unterlizenzieren. § 18 der Softwarenutzungsbedingungen bleibt hiervon unberührt.

§ 6 Lizenzmodelle

1. STAR bietet dem Kunden für die Lizenzierung der Software die folgenden Lizenzmodelle an:

a. „Single User License" Der Kunde darf die Software zur gleichen Zeit nur durch einen Mitarbeiter auf einem Computer nutzen. Auf welchem Computer und durch welchen Mitarbeiter die Nutzung erfolgt, ist dem Kunden freigestellt. Dem Kunden ist es nicht gestattet, die Software auf dem Server eines Netzwerks vorzuhalten, soweit hierdurch eine zeitgleiche Mehrfachnutzung der Software ermöglicht wird. Will der Kunde die Software gleichzeitig durch mehrere Mitarbeiter oder auf mehreren Computern nutzten, muss der Kunde dies vorher schriftlich anzeigen und entgeltlich weitere Lizenzen für die Software bei STAR erwerben.

b. „Floating License" Der Kunde darf die Software zur gleichen Zeit durch mehrere Mitarbeiter auf verschiedenen Computern nutzen. Die Anzahl der gleichzeitig zur Nutzung berechtigten Mitarbeiter ergibt sich aus der Auftragsbestätigung. Durch welchen Mitarbeiter und auf welchen Computern die Nutzung erfolgt, ist dem Kunden freigestellt. Will der Kunde die Zahl der gleichzeitig zur Nutzung berechtigten Mitarbeiter erhöhen, muss der Kunde dies STAR vorher schriftlich anzeigen und entgeltlich weitere Lizenzen für die Software bei STAR erwerben.

c. „Integrated Software License" Die Integrated Software License gilt für Gerätetreiber, Firmware und Embedded Software. Der Kunde darf unter der Integrated Software License lizenzierte Software ausschließlich für den bestimmungsgemäßen Betrieb der gemeinsam mit dieser Software bei STAR erworbenen Hardware nutzen. Der bestimmungsgemäße Betrieb ergibt sich aus der jeweiligen Anwenderdokumentation. Jede anderweitige Nutzung der Software, insbesondere, aber nicht nur, die Nutzung der Software für Hardware, die der Kunde nicht gemeinsam mit der Software bei STAR erworben hat, ist ausgeschlossen.

d. „Development Kit License" Der Kunde darf die Software für das in der Auftragsbestätigung genannte Projekt nutzen. Will der Kunde die Software für andere Projekte nutzten, muss der Kunde dies vorher schriftlich STAR anzeigen und entgeltlich weitere Lizenzen für die Software bei STAR erwerben. Ist in der Auftragsbestätigung trotz der Angabe des Lizenzmodells „Development Kit License" kein Projekt benannt, gilt die Development Kit License für das zur Lizenzvergabe sachnächste Projekt des Kunden. Die unter der Development Kit License lizenzierte Software darf im Rahmen des jeweiligen Projekts

- (i) abweichend von § 5 Abs. 2 dieser Softwarenutzungsbedingungen vom Kunden nicht nur für interne Zwecke, sondern auch für die Zwecke Dritter genutzt werden.
- (ii) abweichend von § 5 Abs. 2 dieser Softwarenutzungsbedingungen zum Zweck der unentgeltlichen oder entgeltlichen Weitergabe vom Kunden entwickelter Software vervielfältigt werden. Wobei die Weitergabe der Software an Dritte nur unter der Bedingung erlaubt ist, dass der Kunde dem Dritten die Datei „Copyright.txt" weitergibt und sich der Dritte verpflichtet, die Einschränkungen aus der Auftragsbestätigung, der Anwenderdokumentation, den Softwarenutzungsbedingungen und den sich aus der Datei „Copyright.txt" ergebenden Einschränkungen bei der Nutzung der Software einzuhalten.
- (iii) abweichend von § 5 Abs. 4 an Dritte unterlizenziert werden, sofern sich der jeweilige Dritte verpflichtet, die Einschränkungen aus der Auftragsbestätigung, der Anwenderdokumentation, diesen Softwarenutzungsbedingungen und den sich aus der Datei „Copyright.txt" ergebenden Einschränkungen bei der Nutzung der Software einzuhalten. Jede Unterlizenz ist im Bestand und Umfang von der Hauptlizenz abhängig.

2. Gerätetreiber, Firmware und Embedded Software werden immer unter der Integrated Software License lizenziert. Im übrigen gilt: Soweit nicht ausdrücklich etwas anderes in der Auftragsbestätigung geregelt ist, erwirbt der Kunde nur eine „Single User License".

§ 7 Sicherungskopie

Zur vertragsgemäßen Nutzung der Software und den darin enthaltenen Datenbeständen gehört die Herstellung einer Sicherungskopie. Die Sicherungskopie darf ausschließlich zu Sicherungszwecken verwendet werden. Der Kunde wird die Sicherungskopie an einem sicheren, für Dritte nicht zugänglichen Ort aufbewahren.

§ 8 Quellcode/Objektcode

1. Die Software wird dem Kunden im Objektcode als Download oder auf maschinenlesbaren Datenträgern überlassen.
2. Soweit in der Auftragsbestätigung nicht ausdrücklich etwas anderes geregelt ist, hat der Kunde keinen Anspruch auf die

Überlassung des Quellcodes oder von Teilen des Quellcodes.

3. Eine Rückübersetzung des Objektcodes ist nur unter Einhaltung der gesetzlichen Beschränkungen gemäß § 69e UrhG zulässig. Weitergehende Rückübersetzungen sind ausgeschlossen.

§ 9 Anwenderdokumentation

1. Der Kunde erhält eine Anwenderdokumentation in druckschriftlicher Form und/oder in elektronischer Form auf maschinenlesbaren Datenträgern oder als Download.

2. Der Kunde darf die Anwenderdokumentation in druckschriftlicher Form nur mit vorheriger schriftlicher Zustimmung von STAR vervielfältigen. Erhält der Kunde die Anwenderdokumentation nur in elektronischer Form, darf der Kunde abweichend von Satz 1 eine Vervielfältigung der Anwenderdokumentation in druckschriftlicher Form für den eigenen internen Gebrauch erstellen.

3. Im Übrigen gelten für die Nutzung der Anwenderdokumentation in elektronischer Form § 5 bis § 7 der Softwarenutzungsbedingungen sinngemäß.

§ 10 Schutz des Lizenzmaterials

1. Der Kunde verpflichtet sich, die im Lizenzmaterial enthaltenen Schutzvermerke wie Copyrightvermerke und andere Rechtsvorbehalte zu beachten, unverändert beizubehalten sowie in der Sicherungskopie in unveränderter Form zu übernehmen.

2. Unbeschadet der auf Grund dieser Softwarenutzungsbedingungen eingeräumten Nutzungsrechte behält STAR alle Rechte am Lizenzmaterial einschließlich aller vom Kunden hergestellten Kopien oder Teilkopien desselben. Das Eigentum des Kunden an maschinenlesbaren Datenträgern und Hardware wird hiervon nicht berührt.

§ 11 Softwarepflege

1. Voraussetzung für die Leistung von Softwarepflege ist der Abschluss eines entgeltlichen Pflegevertrags. Ohne Pflegevertrag hat der Kunde keinen Anspruch auf Softwarepflege.

2. Gegenstand der Softwarepflege ist die Lieferung von Updates der Software, sobald STAR diese im normalen Geschäftsverkehr auch anderen Kunden zur Verfügung stellt.

3. Pflegeverträge haben eine Grundlaufzeit von einem Kalenderjahr beginnend mit dem Tag der Überlassung der jeweiligen Software. Wird der Pflegevertrag nicht spätestens zwei Kalendermonate vor Ablauf der Grundlaufzeit von einer der Vertragsparteien schriftlich gekündigt, verlängert sich der Pflegevertrag um ein weiteres Kalenderjahr. Satz 2 gilt für jede nachfolgende Verlängerung entsprechend.

4. Für die Nutzung von Updates der Software gelten diese Softwarenutzungsbedingungen in ihrer bei Lieferung des jeweiligen Updates gültigen Fassung.

§ 12 Gewährleistung für Rechtsmängel

1. Der Kunde wird STAR unverzüglich schriftlich benachrichtigen, falls ihm gegenüber Ansprüche wegen der Verletzung von Schutzrechten durch das Lizenzmaterial geltend gemacht werden.

2. Wird der Kunde an der vertragsgemäßen Nutzung des Lizenzmaterials durch Schutzrechte Dritter beeinträchtigt, so hat STAR in einem für den Kunden zumutbaren Umfang das Recht, nach der Wahl von STAR und auf Kosten von STAR entsprechende Lizenzen vom Dritten zu erwerben oder das Lizenzmaterial so zu ändern oder auszutauschen, dass es die Schutzrechte des Dritten nicht mehr beeinträchtigt.

3. STAR trifft keine Verpflichtung zur Rechtsmängelgewährleistung, soweit die Verletzung von Schutzrechten Dritter darauf beruht, dass der Kunde das Lizenzmaterial geändert oder bearbeitet hat, oder darauf, dass der Kunde die Software zusammen mit nicht von STAR gelieferter Hardware oder Computerprogrammen nutzt. Dies gilt auch, wenn die Ansprüche des Dritten auf einer Nutzung des Lizenzmaterials durch den Kunden beruhen, nachdem STAR den Kunden darauf

hingewiesen hat, dass die Nutzung wegen eines Anspruchs von Dritten einzustellen ist.

§ 13 Gewährleistung für Sachmängel

1. Die vertragsgemäße Benutzung und die Einsatzbedingungen der Software ergeben sich aus der Anwenderdokumentation. Die Verantwortung für die Auswahl der Software und den Einsatz der Software einschließlich der durch deren Einsatz herbeizuführenden Leistungsergebnisse liegt allein beim Kunden.

2. Für die Software in der dem Kunden überlassenen Fassung gewährleistet STAR die Eignung für den vertragsgemäßen Gebrauch in Übereinstimmung mit der Anwenderdokumentation. Im Fall von erheblichen Abweichungen von der Anwenderdokumentation ist STAR zur Nachbesserung berechtigt und, soweit dies nicht mit unangemessenem Aufwand verbunden ist, auch verpflichtet. Ist dem Kunden das Abwarten auf das nächste Update der Software nicht zumutbar und gelingt es STAR innerhalb einer angemessenen Frist nicht, durch Nachbesserung die erheblichen Abweichungen der Software von der Anwenderdokumentation zu beseitigen oder so zu umgehen, dass dem Kunden der vertragsgemäße Gebrauch der Software ermöglicht wird, kann der Kunde nach seiner Wahl zurücktreten oder stattdessen eine Herabsetzung der Lizenzgebühr gemäß § 441 BGB verlangen.

3. STAR erfüllt die Gewährleistungspflichten während der Üblichen Arbeitszeiten; insoweit entstehen für den Kunden keine Kosten. Soll STAR Gewährleistungspflichten außerhalb der üblichen Arbeitszeiten erfüllen, so ist eine schriftliche Zusatzvereinbarung zum Ausgleich der anfallenden Mehrkosten erforderlich.

4. Der Kunde ist verpflichtet, STAR nachprüfbare Unterlagen über Art und Auftreten von Abweichungen der Software von der Anwenderdokumentation zur Verfügung zu stellen und bei der Eingrenzung von Fehlern mitzuwirken.

5. Offensichtliche Mängel sind STAR innerhalb einer Frist von 2 (zwei) Wochen ab Empfang der Software anzuzeigen; anderenfalls ist die Geltendmachung des Gewährleistungsanspruchs ausgeschlossen. Zur Fristwahrung genügt die rechtzeitige Absendung der Mängelanzeige; die Beweislast hierfür trifft den Kunden. Ist der Kunde Kaufmann, gelten die Regelungen des § 377 HGB.

6. STAR trifft keine Verpflichtung zur Sachmängelgewährleistung, wenn der Kunde die Software geändert oder bearbeitet hat.

7. Ebenfalls trifft STAR keine Verpflichtung zur Sachmängelgewährleistung, wenn der Mangel der Software darauf beruht, dass der Kunde die Software zusammen mit nicht von STAR gelieferter Hardware oder Computerprogrammen nutzt.

8. Erbringt STAR in den Fällen des § 13 Abs. 6 und 7 dieser Softwarenutzungsbedingungen dennoch Arbeiten zur Mängelbeseitigung, hat der Kunde diese Arbeiten nach den zum Zeitpunkt der Ausführung dieser Arbeiten üblichen Sätzen von STAR zu vergüten.

9. Die vorstehenden Einschränkungen der Gewährleistungspflicht gelten nicht in Fällen, in denen STAR eine Garantie für die Beschaffenheit des Lizenzmaterials übernommen oder Mängel arglistig verschwiegen hat.

§ 14 Verjährung von Gewährleistungsansprüchen

Gewährleistungsansprüche verjähren – vorbehaltlich der Regelungen im folgenden Satz und in § 13 Abs. 9 der Softwarenutzungsbedingungen – in einem Jahr ab Ablieferung der Software. Schadensersatz- oder Aufwendungsersatzansprüche des Kunden bleiben hingegen durch die vorstehenden Regelungen unberührt und verjähren innerhalb der gesetzlichen Fristen; für Schadensersatz- und Aufwendungsersatzansprüche gelten die Regelungen in § 15 der Softwarenutzungsbedingungen.

§ 15 Haftung

1. STAR haftet dem Kunden bei der Verletzung wesentlicher Vertragspflichten (sogenannter Kardinalpflichten) auf Aufwendungs- und Schadensersatz (im Folgenden in § 15 der Softwarenutzungsbedingungen gemeinsam **„Schadensersatz"**). Kardinalpflichten sind alle Pflichten, deren Verletzung die Erreichung des Vertragszwecks gefährdet sowie alle Pflichten, deren Erfüllung die ordnungsgemäße Durchführung des Vertrags überhaupt erst ermöglicht und auf deren Einhaltung der Kunde regelmäßig vertrauen darf. Soweit jedoch die Verletzung einer Kardinalpflicht nur leicht fahrlässig geschah und nicht zu einer Verletzung von Leben, Körper oder Gesundheit führte, sind Ansprüche des Kunden auf

Schadensersatz der Höhe nach auf den typischen vorhersehbaren Schaden beschränkt.

2. STAR haftet dem Kunde außerdem nach den Vorschriften des Produkthaftungsgesetzes, in Fällen des Vorsatzes und der groben Fahrlässigkeit, für die Verletzung des Lebens, des Körpers oder der Gesundheit, bei Übernahme einer Garantie sowie in allen anderen Fällen gesetzlich zwingender Haftung, jeweils nach Maßgabe der gesetzlichen Vorschriften.

3. Im Übrigen sind Ansprüche auf Schadensersatz gegen STAR – gleich aus welchem Rechtsgrund, insbesondere wegen Verletzung von Pflichten aus diesem Schuldverhältnis durch STAR, deren gesetzliche Vertreter, Mitarbeiter oder Erfüllungsgehilfen, aus § 311a BGB oder aus unerlaubter Handlung − ausgeschlossen.

4. Soweit nach den vorstehenden Regelungen die Haftung von STAR eingeschränkt oder ausgeschlossen ist, gilt das auch für die persönliche Haftung der gesetzlichen Vertreter, Mitarbeiter und Erfüllungsgehilfen von STAR.

§ 16 Einsatzbedingungen für die Software

1. Die dem Kunde überlassene Software wurde für den Einsatz auf bestimmter Hardware und für das Zusammenwirken mit bestimmten anderen Computerprogrammen entwickelt. Diese Einsatzbedingungen sind im Einzelnen in der jeweiligen Anwenderdokumentation angegeben, die vom Kunden zwingend zu beachten ist. Der Kunde ist verpflichtet, die Software vor dem produktiven Einsatz sorgfältig zu testen. Dies gilt insbesondere, aber nicht nur, wenn die Software in einem sicherheitskritischen Bereich eingesetzt werden soll.

2. Bei der Benutzung der Software ohne Einhaltung der Einsatzbedingungen gemäß § 16 Abs. 1 dieser Softwarenutzungsbedingungen entfällt die Verpflichtung zur Gewährleistung nach § 12 und § 13 dieser Softwarenutzungsbedingungen. Erbringt STAR dennoch Arbeiten zur Mängelbeseitigung, hat der Kunde diese Arbeiten nach den zum Zeitpunkt der Ausführung dieser Arbeiten üblichen Sätzen von STAR zu vergüten.

3. Machen Dritte gegenüber STAR Ansprüche geltend und beruhen diese Ansprüche darauf, dass der Kunde, dessen gesetzliche Vertreter, Mitarbeiter oder Erfüllungsgehilfen die Software schuldhaft ohne Einhaltung der Einsatzbedingungen gemäß § 16 Abs. 1 dieser Softwarenutzungsbedingungen genutzt haben, wird der Kunde STAR von diesen Ansprüchen des Dritten freistellen und STAR alle in diesem Zusammenhang entstehenden Schäden ersetzen. STAR wird den Kunden unverzüglich schriftlich benachrichtigen, falls STAR gegenüber Ansprüche im Sinne von Satz 1 geltend gemacht werden.

§ 17 Testphase

1. Der Kunde darf die Software vor der produktiven Nutzung kostenlos testen, wenn dies ausdrücklich in der Auftragsbestätigung vorgesehen ist. In diesem Fall ergeben sich auch die einzelnen Modalitäten der Testphase (Dauer, etc.) aus der Auftragsbestätigung.

2. In jedem Fall endet die Testphase, wenn der Kunde mit der produktiven Nutzung der Software beginnt. Der Kunde ist verpflichtet, STAR den Beginn der produktiven Nutzung schriftlich anzuzeigen.

§ 18 Weitergabe des Lizenzmaterials

1. Der Kunde ist berechtigt, das Lizenzmaterial im Originalzustand zusammen mit einer Kopie dieser Softwarenutzungsbedingungen und der Auftragsbestätigung an einen Dritten abzugeben, soweit der Kunde STAR den Namen und die vollständige Adresse des Dritten schriftlich mitteilt und sich der Dritte gegenüber STAR verpflichtet, die Einschränkungen aus der Auftragsbestätigung, der Anwenderdokumentation und diesen Softwarenutzungsbedingungen bei der Nutzung der Software einzuhalten. Diese Berechtigung zur Weitergabe des Lizenzmaterials erstreckt sich nicht auf geänderte oder bearbeitete Fassungen des Lizenzmaterials.

2. Mit der Weitergabe des Lizenzmaterials unter Einhaltung der Voraussetzungen gemäß § 18 Abs. 1 der Softwarenutzungsbedingungen geht die Berechtigung zur Nutzung des Lizenzmaterials auf den Dritten über, der damit im Sinne der Softwarenutzungsbedingungen an die Stelle des Kunden tritt. Zugleich erlischt die Berechtigung des Kunden, die Software zu nutzen.

3. Mit der Weitergabe des Lizenzmaterials hat der Kunde alle Kopien und Teilkopien des Lizenzmaterials sowie geänderte oder bearbeitete Fassungen des Lizenzmaterials umgehend und vollständig zu löschen oder auf andere Weise zu vernichten. Dies

gilt auch für die Sicherungskopie gemäß § 7 der Softwarenutzungsbedingungen.

4. § 18 Abs. 1 bis 3 der Softwarenutzungsbedingungen gelten auch, wenn die Weitergabe in einer zeitweiser Überlassung besteht. Die Vermietung des Lizenzmaterials oder von Teilen davon ist jedoch ausgeschlossen.

5. Für nachfolgende Weitergaben des Lizenzmaterials gelten § 18 Abs. 1 bis 4 der Softwarenutzungsbedingungen entsprechend.

6. Die vorstehenden Regelungen dieses § 18 der Softwarenutzungsbedingungen gelten nicht für Software, die unter dem Lizenzmodell „Development Kit License" lizenziert wird. Die Weitergabe dieser Software an Dritte ist nur unter den in § 6 Abs. 1 lit. d der Softwarenutzungsbedingungen geregelten Voraussetzungen zulässig.

§ 19 Lizenzgebühr/ Pflegegebühren

1. Der Kunde zahlt die in der Auftragsbestätigung ausgewiesenen Lizenz- und Pflegegebühren. Die in der Auftragsbestätigung aufgeführten Beträge enthalten keine Umsatzsteuer, diese wird gesondert mit dem jeweils gültigen Satz in Rechnung gestellt.

2. Im Übrigen gelten für die Zahlung der Lizenz- und der Pflegegebühren die Allgemeinen Verkaufsbedingungen von STAR, die dem Kunden vor Vertragsschluss zur Verfügung gestellt worden sind.

§ 20 Demosoftware

1. STAR bietet neben der entgeltlichen Software auch Demosoftware im Objektcode auf maschinenlesbaren Datenträgern oder als Download an. Der Kunde hat keinen Anspruch auf die Überlassung von Demosoftware, auf die Überlassung des Quellcodes oder von Teilen des Quellcodes von Demosoftware oder auf die Überlassung von Updates von Demosoftware.

2. Der Vertrag zwischen dem Kunden und STAR über die Überlassung von Demosoftware kommt durch die Ingebrauchnahme der Demosoftware durch den Kunden zustande.

3. Der Kunde erhält für die Dauer des Bestehens etwaiger Schutzrechte ein nicht ausschließliches Recht, die Demosoftware für die in der Anwenderdokumentation vorgesehenen Zwecke intern zu nutzen. Nutzung der Demosoftware ist dabei jede dauerhafte oder vorübergehende ganze oder teilweise Vervielfältigung der Demosoftware durch Speichern, Laden, Ablaufen oder Anzeigen zum Zweck der Ausführung der Demosoftware; jede anderweitige Vervielfältigung der Demosoftware ist mit Ausnahme der Anfertigung einer Sicherungskopie (§ 7 der Softwarenutzungsbedingungen) ausgeschlossen. Ausgeschlossen ist insbesondere, aber nicht nur, die Vervielfältigung der Demosoftware zum Zweck der entgeltlichen oder unentgeltlichen Weitergabe an Dritte.

4. Der Kunde hat kein Recht, die Demosoftware zu ändern oder zu bearbeiten. Eine Rückübersetzung des Objektcodes ist ausschließlich unter den gesetzlichen Beschränkungen gemäß § 69e UrhG zulässig. Weitergehende Rückübersetzungen sind ausgeschlossen.

5. Der Kunde darf hinsichtlich der ihm an der Demosoftware eingeräumten Rechte Dritten keine Rechte einräumen. Ebenfalls darf der Kunde die ihm an der Demosoftware eingeräumten Rechte nicht an Dritte unterlizenzieren.

6. Für Rechts- und Sachmängel der Demosoftware und der jeweiligen Anwenderdokumentation haftet STAR nur, wenn STAR einen Mangel arglistig verschweigt. Außerhalb der Mängelgewährleistung haftet STAR nur für Vorsatz und grobe Fahrlässigkeit. Im Übrigen ist die Haftung von STAR ausgeschlossen. Soweit nach den vorstehenden Regelungen die Haftung von STAR eingeschränkt oder ausgeschlossen ist, gilt das auch für die persönliche Haftung der gesetzlichen Vertreter, Mitarbeiter und Erfüllungsgehilfen von STAR.

7. Die dem Kunde überlassene Demosoftware wurde für den Einsatz auf bestimmter Hardware und für das Zusammenwirken mit bestimmten anderen Computerprogrammen entwickelt. Diese Einsatzbedingungen sind im Einzelnen in der jeweiligen Anwenderdokumentation angegeben. Machen Dritte gegenüber STAR Ansprüche geltend und beruhen diese Ansprüche darauf, dass der Kunde, dessen gesetzliche Vertreter, Mitarbeiter oder Erfüllungsgehilfen die Demosoftware schuldhaft ohne Einhaltung der Einsatzbedingungen gemäß § 20 Abs. 7 Satz 2 der Softwarenutzungsbedingungen genutzt haben, wird der Kunde STAR von diesen Ansprüchen des Dritten freistellen und STAR alle in diesem Zusammenhang entstehenden Schäden

ersetzen. STAR wird den Kunden unverzüglich schriftlich benachrichtigen, falls STAR gegenüber derartige Ansprüche geltend gemacht werden.

8. Auf die Überlassung der Demosoftware sind im Übrigen sinngemäß folgende Bestimmungen der Softwarenutzungsbedingungen anwendbar: § 7, § 9 (mit der Änderung, dass § 9 Abs. 3 auf diesen § 20 verweist), § 10, § 22.

§ 21 Geheimhaltung

1. Die Geheimhaltungspflichten beider Vertragsparteien ergeben sich aus einer gesonderten Geheimhaltungsvereinbarung. Haben die Vertragsparteien keine gesonderte Geheimhaltungsvereinbarung abgeschlossen, gelten die nachfolgenden Abs. 2 bis 4.

2. Die Vertragsparteien verpflichten sich, sämtliche ihnen im Zusammenhang mit diesem Vertrag zugänglich werdenden vertraulichen Informationen unbefristet geheim zu halten. Vertrauliche Informationen sind Informationen, die entweder durch eine Vertragspartei als geschützt oder vertraulich markiert oder in anderer Weise schriftlich gekennzeichnet sind, oder Informationen, die gemäß den Umständen ihrer Offenlegung von der empfangenen Vertragspartei vernünftigerweise als vertraulich erkennbar sind. Zu den vertraulichen Informationen gehört insbesondere, aber nicht nur, das Lizenzmaterial.

3. Die vorstehende Geheimhaltungsverpflichtung gilt nicht, wenn und soweit die jeweiligen Informationen nachweislich (i) allgemein bekannt sind oder ohne Verschulden einer Vertragspartei und ohne Verstoß gegen diese Geheimhaltungsverpflichtung allgemein bekannt werden, (ii) Stand der Technik sind oder werden, (iii) der empfangenden Vertragspartei zum Zeitpunkt der Übermittlung bereits bekannt sind, was durch Unterlagen bewiesen werden muss, die eine solche Kenntnis belegen, (iv) der empfangenden Vertragspartei von einem Dritten rechtmäßig bekannt oder zugänglich gemacht wurden oder werden, (iv) aufgrund gesetzlicher Vorschriften oder vollstreckbarer behördlicher Verfügungen oder gerichtlicher Entscheidungen offengelegt werden müssen. Die Beweislast für das Vorliegen eines Ausnahmetatbestandes trägt der jeweilige Informationsempfänger. In jedem Fall ist die jeweils betroffene Vertragspartei rechtzeitig vor der Weitergabe der Informationen an Dritte zu informieren, soweit dies möglich ist.

4. Beide Parteien werden angemessene Vorkehrungen zur Sicherung der vertraulichen Informationen der jeweils anderen Partei treffen. Beide Parteien werden vertrauliche Informationen der jeweils anderen Partei Organen, Mitarbeitern, Beratern oder Subunternehmern nur offen legen vorbehaltlich dieser Vertraulichkeitsverpflichtung, der die Empfänger dann entsprechend zu unterwerfen sind.

§ 22 Sonstiges

1. Sollten einzelne Bestimmungen der Softwarenutzungsbedingungen unwirksam sein oder werden, so wird die Wirksamkeit der übrigen Bestimmungen hierdurch nicht berührt.

2. STAR wird personenbezogene Daten des Kunden zum Zweck der Abwicklung der Softwareüberlassung in einer EDV-Anlage speichern und automatisch verarbeiten. Zu darüber hinausgehenden Zwecken werden die Daten nicht verwendet. Eine Weitergabe der Daten an Dritte erfolgt nicht.

3. Es gilt das Recht der Bundesrepublik Deutschland. Das UN-Übereinkommen über Verträge über den internationalen Warenkauf findet keine Anwendung.

4. Ist der Kunde Kaufmann, juristische Person des öffentlichen Rechts oder öffentlich-rechtliches Sondervermögen, ist ausschließlicher Gerichtsstand für alle Streitigkeiten aus oder im Zusammenhang mit diesem Vertrag Stuttgart. STAR ist berechtigt, den Kunden wahlweise an dessen allgemeinem Gerichtsstand zu verklagen.

License of the used Serilog Library / Lizenz der verwendeten Serilog Bibliothek

(I) Apache License Version 2.0

Apache

License
Version
2.0,
January
2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License.

Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License.

Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent

infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution.

You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works hat You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions.

Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks.

This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty.

Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability.

In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the

possibility of such damages.

9. Accepting Warranty or Additional Liability.

While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

License of the used NETStandard Library / Lizenz der verwendeten NETStandard Bibliothek

(II) MIT License

The MIT License (MIT)

Copyright (c) 2007 James Newton-King

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

License of the used PacketDotNet Library / Lizenz der verwendeten PacketDotNet Bibliothek

(III) Mozilla Public License Version 2.0

1. Definitions

1.1. “Contributor”
means each individual or legal entity that creates, contributes to the creation of, or owns Covered Software.

1.2. “Contributor Version”
means the combination of the Contributions of others (if any) used by a Contributor and that particular Contributor's Contribution.

1.3. “Contribution”
means Covered Software of a particular Contributor.

1.4. “Covered Software”
means Source Code Form to which the initial Contributor has attached the notice in Exhibit A, the Executable Form of such Source Code Form, and Modifications of such Source Code Form, in each case including portions thereof.

1.5. “Incompatible With Secondary Licenses”
means

- **(a)** that the initial Contributor has attached the notice described in Exhibit B to the Covered Software; or
- **(b)** that the Covered Software was made available under the terms of version 1.1 or earlier of the License, but not also under the terms of a Secondary License.

1.6. “Executable Form”
means any form of the work other than Source Code Form.

1.7. “Larger Work”
means a work that combines Covered Software with other material, in a separate file or files, that is not Covered Software.

1.8. “License”
means this document.

1.9. “Licensable”
means having the right to grant, to the maximum extent possible, whether at the time of the initial grant or subsequently, any and all of the rights conveyed by this License.

1.10. “Modifications”
means any of the following:

- **(a)** any file in Source Code Form that results from an addition to, deletion from, or modification of the contents of Covered Software; or
- **(b)** any new file in Source Code Form that contains any Covered Software.

1.11. “Patent Claims” of a Contributor
means any patent claim(s), including without limitation, method, process, and apparatus claims, in any patent Licensable by such Contributor that would be infringed, but for the grant of the License, by the making, using, selling, offering for sale, having made, import, or transfer of either its Contributions or its Contributor Version.

1.12. “Secondary License”
means either the GNU General Public License, Version 2.0, the GNU Lesser General Public License, Version 2.1, the GNU Affero General Public License, Version 3.0, or any later versions of those licenses.

1.13. “Source Code Form”
means the form of the work preferred for making modifications.

1.14. “You” (or “Your”)
means an individual or a legal entity exercising rights under this License. For legal entities, “You” includes any entity that controls, is controlled by, or is under common control with You. For purposes of this definition, “control” means **(a)** the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or **(b)** ownership of more than fifty percent (50%) of the outstanding shares or beneficial ownership of such entity.

2. License Grants and Conditions

2.1. Grants

Each Contributor hereby grants You a world-wide, royalty-free, non-exclusive license:

- **(a)** under intellectual property rights (other than patent or trademark) Licensable by such Contributor to use, reproduce, make available, modify, display, perform, distribute, and otherwise exploit its Contributions, either on an unmodified basis, with Modifications, or as part of a Larger Work; and
- **(b)** under Patent Claims of such Contributor to make, use, sell, offer for sale, have made, import, and otherwise transfer either its Contributions or its Contributor Version.

2.2. Effective Date

The licenses granted in Section 2.1 with respect to any Contribution become effective for each Contribution on the date the Contributor first distributes such Contribution.

2.3. Limitations on Grant Scope

The licenses granted in this Section 2 are the only rights granted under this License. No additional rights or licenses will be implied from the distribution or licensing of Covered Software under this License. Notwithstanding Section 2.1(b) above, no patent license is granted by a Contributor:

- **(a)** for any code that a Contributor has removed from Covered Software; or
- **(b)** for infringements caused by: **(i)** Your and any other third party's modifications of Covered Software, or **(ii)** the combination of its Contributions with other software (except as part of its Contributor Version); or
- **(c)** under Patent Claims infringed by Covered Software in the absence of its Contributions.

This License does not grant any rights in the trademarks, service marks, or logos of any Contributor (except as may be necessary to comply with the notice requirements in Section 3.4).

2.4. Subsequent Licenses

No Contributor makes additional grants as a result of Your choice to distribute the Covered Software under a subsequent version of this License (see Section 10.2) or under the terms of a Secondary License (if permitted under the terms of Section 3.3).

2.5. Representation

Each Contributor represents that the Contributor believes its Contributions are its original creation(s) or it has sufficient rights to grant the rights to its Contributions conveyed by this License.

2.6. Fair Use

This License is not intended to limit any rights You have under applicable copyright doctrines of fair use, fair dealing, or other equivalents.

2.7. Conditions

Sections 3.1, 3.2, 3.3, and 3.4 are conditions of the licenses granted in Section 2.1.

3. Responsibilities

3.1. Distribution of Source Form

All distribution of Covered Software in Source Code Form, including any Modifications that You create or to which You contribute, must be under the terms of this License. You must inform recipients that the Source Code Form of the Covered Software is governed by the terms of this License, and how they can obtain a copy of this License. You may not attempt to alter or restrict the recipients' rights in the Source Code Form.

3.2. Distribution of Executable Form

If You distribute Covered Software in Executable Form then:

- **(a)** such Covered Software must also be made available in Source Code Form, as described in Section 3.1, and You must inform recipients of the Executable Form how they can obtain a copy of such Source Code Form by reasonable means in a timely manner, at a charge no more than the cost of distribution to the recipient; and

- **(b)** You may distribute such Executable Form under the terms of this License, or sublicense it under different terms, provided that the license for the Executable Form does not attempt to limit or alter the recipients' rights in the Source Code Form under this License.

3.3. Distribution of a Larger Work

You may create and distribute a Larger Work under terms of Your choice, provided that You also comply with the requirements of this License for the Covered Software. If the Larger Work is a combination of Covered Software with a work governed by one or

more Secondary Licenses, and the Covered Software is not Incompatible With Secondary Licenses, this License permits You to additionally distribute such Covered Software under the terms of such Secondary License(s), so that the recipient of the Larger Work may, at their option, further distribute the Covered Software under the terms of either this License or such Secondary License(s).

3.4. Notices

You may not remove or alter the substance of any license notices (including copyright notices, patent notices, disclaimers of warranty, or limitations of liability) contained within the Source Code Form of the Covered Software, except that You may alter any license notices to the extent required to remedy known factual inaccuracies.

3.5. Application of Additional Terms

You may choose to offer, and to charge a fee for, warranty, support, indemnity or liability obligations to one or more recipients of Covered Software. However, You may do so only on Your own behalf, and not on behalf of any Contributor. You must make it absolutely clear that any such warranty, support, indemnity, or liability obligation is offered by You alone, and You hereby agree to indemnify every Contributor for any liability incurred by such Contributor as a result of warranty, support, indemnity or liability terms You offer. You may include additional disclaimers of warranty and limitations of liability specific to any jurisdiction.

4. Inability to Comply Due to Statute or Regulation

If it is impossible for You to comply with any of the terms of this License with respect to some or all of the Covered Software due to statute, judicial order, or regulation then You must: **(a)** comply with the terms of this License to the maximum extent possible; and **(b)** describe the limitations and the code they affect. Such description must be placed in a text file included with all distributions of the Covered Software under this License. Except to the extent prohibited by statute or regulation, such description must be sufficiently detailed for a recipient of ordinary skill to be able to understand it.

5. Termination

5.1. The rights granted under this License will terminate automatically if You fail to comply with any of its terms. However, if You become compliant, then the rights granted under this License from a particular Contributor are reinstated **(a)** provisionally, unless and until such Contributor explicitly and finally terminates Your grants, and **(b)** on an ongoing basis, if such Contributor fails to notify You of the non-compliance by some reasonable means prior to 60 days after You have come back into compliance. Moreover, Your grants from a particular Contributor are reinstated on an ongoing basis if such Contributor notifies You of the non-compliance by some reasonable means, this is the first time You have received notice of non-compliance with this License from such Contributor, and You become compliant prior to 30 days after Your receipt of the notice.

5.2. If You initiate litigation against any entity by asserting a patent infringement claim (excluding declaratory judgment actions, counter-claims, and cross-claims) alleging that a Contributor Version directly or indirectly infringes any patent, then the rights granted to You by any and all Contributors for the Covered Software under Section 2.1 of this License shall terminate.

5.3. In the event of termination under Sections 5.1 or 5.2 above, all end user license agreements (excluding distributors and resellers) which have been validly granted by You or Your distributors under this License prior to termination shall survive termination.

6. Disclaimer of Warranty

> Covered Software is provided under this License on an “as is” basis, without warranty of any kind, either expressed, implied, or statutory, including, without limitation, warranties that the Covered Software is free of defects, merchantable, fit for a particular purpose or non-infringing. The entire risk as to the quality and performance of the Covered Software is with You. Should any Covered Software prove defective in any respect, You (not any Contributor) assume the cost of any necessary servicing, repair, or correction. This disclaimer of warranty constitutes an essential part of this License. No use of any Covered Software is authorized under this License except under this disclaimer.

7. Limitation of Liability

> Under no circumstances and under no legal theory, whether tort (including negligence), contract, or otherwise, shall any

> Contributor, or anyone who distributes Covered Software as permitted above, be liable to You for any direct, indirect, special, incidental, or consequential damages of any character including, without limitation, damages for lost profits, loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses, even if such party shall have been informed of the possibility of such damages. This limitation of liability shall not apply to liability for death or personal injury resulting from such party's negligence to the extent applicable law prohibits such limitation. Some jurisdictions do not allow the exclusion or limitation of incidental or consequential damages, so this exclusion and limitation may not apply to You.

8. Litigation

Any litigation relating to this License may be brought only in the courts of a jurisdiction where the defendant maintains its principal place of business and such litigation shall be governed by laws of that jurisdiction, without reference to its conflict-of-law provisions. Nothing in this Section shall prevent a party's ability to bring cross-claims or counter-claims.

9. Miscellaneous

This License represents the complete agreement concerning the subject matter hereof. If any provision of this License is held to be unenforceable, such provision shall be reformed only to the extent necessary to make it enforceable. Any law or regulation which provides that the language of a contract shall be construed against the drafter shall not be used to construe this License against a Contributor.

10. Versions of the License

10.1. New Versions

Mozilla Foundation is the license steward. Except as provided in Section 10.3, no one other than the license steward has the right to modify or publish new versions of this License. Each version will be given a distinguishing version number.

10.2. Effect of New Versions

You may distribute the Covered Software under the terms of the version of the License under which You originally received the Covered Software, or under the terms of any subsequent version published by the license steward.

10.3. Modified Versions

If you create software not governed by this License, and you want to create a new license for such software, you may create and use a modified version of this License if you rename the license and remove any references to the name of the license steward (except to note that such modified license differs from this License).

10.4. Distributing Source Code Form that is Incompatible With Secondary Licenses

If You choose to distribute Source Code Form that is Incompatible With Secondary Licenses under the terms of this version of the License, the notice described in Exhibit B of this License must be attached.

Exhibit A - Source Code Form License Notice

```
This Source Code Form is subject to the terms of the Mozilla Public
License, v. 2.0. If a copy of the MPL was not distributed with this
file, You can obtain one at http://mozilla.org/MPL/2.0/.
```

If it is not possible or desirable to put the notice in a particular file, then You may include the notice in a location (such as a LICENSE file in a relevant directory) where a recipient would be likely to look for such a notice.

You may add additional accurate notices of copyright ownership.

Exhibit B - “Incompatible With Secondary Licenses” Notice

```
This Source Code Form is "Incompatible With Secondary Licenses", as
defined by the Mozilla Public License, v. 2.0.
```