DocInterfaceControl

(iID® reader library)

Product:	microsensys iID®3000 PRO RFID interfaces microsensys iID®4000 UHF RFID interfaces
Product Code:	**-**
Revision:	**2.0**
Date:	2026-06-10

API - Definition for Windows® PC, macOS and Linux

iID® reader library is a .NETStandard library implementation supporting microsensys RFID interfaces. This document describes the software API provided by this library.

This documentation describes the whole software API valid for many microsensys RFID interfaces supporting different functionalities.
A lot of applications are based on standard DOC RFID read/write operations, where we recommend usage of *DOCInterfaceControl*. Within more specific applications usage of TAG-specific functions could be required.

Class name:	**iIDReaderLibrary.DocInterfaceControl**
Tested devices:	MICROSOFT® Windows 10/64bit, 11/64bit
Supported host interface:	RS232 serial, USB, Bluetooth SPP, BluetoothLE SPP, LAN
Version:	2.0
Release date:	2026-06-10
Dependencies:	.NETStandard v2.0, System.IO.Ports v6.0, InTheHand.BluetoothLE v4.0.44
NuGet Package-ID:	Microsensys.iIDReaderLibrary.DocInterfaceControl
NuGet-Link:	https://www.nuget.org/packages/Microsensys.iIDReaderLibrary.DocInterfaceControl/

Content

Short history

This chapter includes a short history of modifications of *DocInterfaceControl*.

Date	Reason	Modification	Release Date / Version	FileName
2021-03	- first release		1.0	official NuGet release
2026-06	- Integration BluetoothLE		2.0	

Flow diagram – How to use DocInterfaceControl (iID® reader library)

Initialization

Constructors

This chapter includes all the available constructors, which are necessary for initialization of the class. All of them provide details on host-side peripheral interface to be used.

public DocInterfaceControl(InterfaceCommunicationSettings _commSettings, int _interfaceType, int _protocolType)

Parameter :
_commSettings – instance of *InterfaceCommunicationSettings* containing host-side peripheral interface details (for example name of serial port, where RFID interface is connected)
_interfaceType – RFID reader interface type value (see *InterfaceTypeEnum*)
_protocolType – protocol to be used for communication with RFID reader (see *ProtocolTypeEnum*)
Result :
success – instance of *DocInterfaceControl*
error - Exception
Description :
Initializes *DocInterfaceControl* class using peripheral interface parameters, interface type and protocol type.

public DocInterfaceControl(InterfaceCommunicationSettings _commSettings, int _interfaceType)

Parameter :
_commSettings – instance of *InterfaceCommunicationSettings* containing host-side peripheral interface details (for example name of serial port, where RFID interface is connected)
_interfaceType – RFID reader interface type value (see *InterfaceTypeEnum*)
Result :
success – instance of *DocInterfaceControl*
error - Exception
Description :
Initializes *DocInterfaceControl* class using peripheral interface parameters, interface type and default protocol type based on selected interface type.

public DocInterfaceControl(ICommunicationHandler _commHandler, int _interfaceType, int _protocolType)

Parameter :
_commHandler – instance of *ICommunicationHandler* interface representing the host-side peripheral interface to communicate with (already initialized)
_interfaceType – RFID reader interface type value (see *InterfaceTypeEnum*)
_protocolType – protocol to be used for communication with RFID reader (see *ProtocolTypeEnum*)
Result :
success – instance of *DocInterfaceControl*
error - Exception
Description :
Initializes *DocInterfaceControl* class using an already initialized instance of *ICommunicationHandler* and new interface type and protocol type. This may be used for RFID readers that support multiple interface or protocol types.

public DocInterfaceControl(ICommunicationHandler _commHandler, int _interfaceType)

Parameter :

_commHandler – instance of *ICommunicationHandler* interface representing the host-side peripheral interface to communicate with (already initialized)
_interfaceType – RFID reader interface type value (see *InterfaceTypeEnum*)

Result :

success – instance of *DocInterfaceControl*
error - Exception

Description :

Initializes *DocInterfaceControl* class using an already initialized instance of *ICommunicationHandler* and new interface type and default protocol type based on selected interface type. This may be used for RFID readers that support multiple interface or protocol types.

Initialization functions

This chapter includes all the functions to handle the interface communication initialization, as well as how to close the interface communication.

public bool Initialize()

Parameter :
-
Result :
success – returns true after the communication interface is open and communication with reader is established.
error – returns “false”.
Description :
Initializes interface communication, opening the communication port and trying to communicate with the reader using the defined peripheral interface parameters, interface type and protocol type.

public async Task<bool> InitializeAsync()

Parameter :
-
Result :
success – returns true after the communication interface is open and communication with reader is established.
error – returns “false”.
Description :
Asynchronous version of *Initialize()* function. This function may be awaited upon.

public void StartInitialize()

Parameter :
-
Result :
-
Description :
Starts the initialization process in a separate thread. This function completes without blocking and waiting for the process result. It initializes interface communication, opening the communication port and trying to communicate with the reader using the defined peripheral interface parameters, interface type and protocol type. Once the process is finished, the result is provided using the *InitializeCompleted* event

Common functions

Reader functions

This chapter describes functions available for communication and configuration of the RFID reader without transponder communication.

public ReaderIDInfo ReadReaderID()

Parameter :
-
Result :
success – returns instance of *ReaderIDInfo* representing information received by reader.
error – returns *null* or *Exception*.
Description :
Communicates with the reader and reads the information. This function blocks the execution until the answer received is or for a maximum of 1000ms.

public ReaderIDInfo ReadReaderID(int _maxScanTimeMs)

Parameter :
_maxScanTimeMs – maximum time in milliseconds that the function will try to read the information from reader.
Result :
success – returns instance of *ReaderIDInfo* representing information received by reader.
error – returns *null* or *Exception*.
Description :
Communicates with the reader and reads the information. This function blocks the execution until the answer received is or for a maximum of time provided by parameter.

RFID functions – blocking

The following chapter describes functions, which are hardware independent and abstracted from different transponder systems based on system-specific functions described in the next chapters. Using these functions is more comfortable, but sometimes also time-expensive, so in time-critical processes the usage of system-specific functions is recommended.

These functions encapsulate reader, frequency, and tag-specific functions to abstract functions. If access to TAG-specific functions is required, you may need to use specific functions instead or additional to these.

public object Identify()

Parameter :

-

Result :

success – returns instance object representing the TAG found. Depending on the configured interface type a different type of object is returned.

HF (1356): Instance of *HfScanResultInfo*

UHF (868): Instance of *UhfScanResultInfo*

error – returns *null* or *Exception*.

Description :

Scans for a TAG near antenna interface. This function blocks the execution until a TAG is found, or for a maximum of 1000ms.

public object Identify(int _maxScanTimeMs)

Parameter :

_maxScanTimeMs – maximum time in milliseconds that the function will try to read the information from reader.

Result :

success – returns instance object representing the TAG found. Depending on the configured interface type a different type of object is returned.

HF (1356): Instance of *HfScanResultInfo*

UHF (868): Instance of *UhfScanResultInfo*

error – returns *null* or *Exception*.

Description :

Scans for a TAG near antenna interface. This function blocks the execution until a TAG is found, or for a maximum of time provided by parameter.

public byte[] ReadBytes(byte[] _tagID, int _page, int _from, int _length)

Parameter :

_tagID – byte array representing the ID of the transponder to read memory from. The value to pass to this function can be obtained using *Identify* function. maximum time in milliseconds that the process will try to read the information from reader.
_page – memory page/bank to be read (only supported by some transponder types)
_from – first memory position to be read (being 0 the first byte of the selected memory)
_length – number of bytes to be read.

Result :

success – returns byte array containing the read data.
error – returns *null* or *Exception*.

Description :

Reads data from transponder with ID provided by *_tagID*. This function blocks the execution until data is successfully read, or for a maximum of 1000ms.

public byte[] ReadBytes(byte[] _tagID, int _page, int _from, int _length, int _maxScanTimeMs)

Parameter :

_tagID – byte array representing the ID of the transponder to read memory from. The value to pass to this function can be obtained using *Identify* function. maximum time in milliseconds that the process will try to read the information from reader.
_page – memory page/bank to be read (only supported by some transponder types)
_from – first memory position to be read (being 0 the first byte of the selected memory)
_length – number of bytes to be read.
_maxScanTimeMs – maximum time in milliseconds that the function will try to read the data.

Result :

success – returns byte array containing the read data.
error – returns *null* or *Exception*.

Description :

Reads data from transponder memory with ID provided by *_tagID*. This function blocks the execution until data is successfully read, or for a maximum of time provided by parameter.

public bool WriteBytes(byte[] _tagID, int _page, int _from, byte[] _data, bool _lock)

Parameter :

_tagID – byte array representing the ID of the transponder to read memory from. The value to pass to this function can be obtained using *Identify* function. maximum time in milliseconds that the process will try to read the information from reader.
_page – memory page/bank to be read (only supported by some transponder types)
_from – first memory position to be read (being 0 the first byte of the selected memory)
_data – data to write to transponder memory.
_lock – if set to true, the memory blocks will be locked after writing (Attention: locking the blocks is irreversible and blockwise! See transponder chip documentation for more information)

Result :

success – returns *true* if data successfully written.
error – returns *false* or *Exception*.

Description :

Writes data into transponder memory with ID provided by *_tagID*. This function blocks the execution until data is successfully read, or for a maximum of 1000ms.

public bool WriteBytes(byte[] _tagID, int _page, int _from, byte[] _data, bool _lock, int _maxScanTimeMs)

Parameter :

_tagID – byte array representing the ID of the transponder to read memory from. The value to pass to this function can be obtained using *Identify* function. maximum time in milliseconds that the process will try to read the information from reader.
_page – memory page/bank to be read (only supported by some transponder types)
_from – first memory position to be read (being 0 the first byte of the selected memory)
_data – data to write to transponder memory.
_lock – if set to true, the memory blocks will be locked after writing (Attention: locking the blocks is irreversible and blockwise! See transponder chip documentation for more information)
_maxScanTimeMs – maximum time in milliseconds that the function will try to write the data.

Result :

success – returns *true* if data successfully written.
error – returns *null* or *Exception*.

Description :

Writes data into transponder memory with ID provided by *_tagID*. This function blocks the execution until data is successfully read, or for a maximum of time provided by parameter.

public TELIDSensorResultInfo GetSensorData(int _sensorType)

Parameter :

_sensorType – this parameter selects the TELID® sensor type to search for.
Default values:
HF (1356): 0xFC – searches for all supported TELID®200 types
UHF (868): 0x00 – searches for all supported TELID®400 types

Result :

success – returns instance of *TELIDSensorInfo* representing the found TELID® and the measured value.
error – returns *null* or *Exception*.

Description :

Scans for a TELID® sensor transponder and performs a measurement. This function blocks the execution until data is successfully read, or for a maximum of 1000ms.

public TELIDSensorResultInfo GetSensorData(int _sensorType, int _maxScanTimeMs)

Parameter :

_sensorType – this parameter selects the TELID® sensor type to search for.
Default values:
HF (1356): 0xFC – searches for all supported TELID®200 types
UHF (868): 0x00 – searches for all supported TELID®400 types
_maxScanTimeMs – maximum time in milliseconds that the function will try search for a TELID® sensor transponder.

Result :

success – returns instance of *TELIDSensorInfo* representing the found TELID® and the measured value.
error – returns *null* or *Exception*.

Description :

Scans for a TELID® sensor transponder and performs a measurement. This function blocks the execution until data is successfully read, or for a maximum of time provided by parameter.

RFID functions – asynchronous

The following chapter describes asynchronous versions of functions above, which are hardware independent and abstracted from different transponder systems based on system-specific functions described in the next chapters. Using these functions is more comfortable, but sometimes also time-expensive, so in time-critical processes the usage of system-specific functions is recommended.

These functions encapsulate reader, frequency, and tag-specific functions to abstract functions. If access to TAG-specific functions is required, you may need to use specific functions instead or additional to these.

public async Task<object> IdentifyAsync()

Parameter :
-
Result :
success – returns instance object representing the TAG found. Depending on the configured interface type a different type of object is returned.
HF (1356): Instance of *HfScanResultInfo*
UHF (868): Instance of *UhfScanResultInfo*
error – returns *null* or *Exception*.
Description :
Scans for a TAG near antenna interface. This function blocks the execution until a TAG is found, or for a maximum of 1000ms.

public async Task<object> IdentifyAsync(int _maxScanTimeMs)

Parameter :
_maxScanTimeMs – maximum time in milliseconds that the function will try to read the information from reader.
Result :
success – returns instance object representing the TAG found. Depending on the configured interface type a different type of object is returned.
HF (1356): Instance of *HfScanResultInfo*
UHF (868): Instance of *UhfScanResultInfo*
error – returns *null* or *Exception*.
Description :
Scans for a TAG near antenna interface. This function blocks the execution until a TAG is found, or for a maximum of time provided by parameter.

public async Task<byte[]> ReadBytesAsync(byte[] _tagID, int _page, int _from, int _length)

Parameter :

_tagID – byte array representing the ID of the transponder to read memory from. The value to pass to this function can be obtained using *Identify* function. maximum time in milliseconds that the process will try to read the information from reader.
_page – memory page/bank to be read (only supported by some transponder types)
_from – first memory position to be read (being 0 the first byte of the selected memory)
_length – number of bytes to be read.

Result :

success – returns byte array containing the read data.
error – returns *null* or *Exception*.

Description :

Reads data from transponder with ID provided by *_tagID*. This function blocks the execution until data is successfully read, or for a maximum of 1000ms.

public async Task<byte[]> ReadBytesAsync(byte[] _tagID, int _page, int _from, int _length, int _maxScanTimeMs)

Parameter :

_tagID – byte array representing the ID of the transponder to read memory from. The value to pass to this function can be obtained using *Identify* function. maximum time in milliseconds that the process will try to read the information from reader.
_page – memory page/bank to be read (only supported by some transponder types)
_from – first memory position to be read (being 0 the first byte of the selected memory)
_length – number of bytes to be read.
_maxScanTimeMs – maximum time in milliseconds that the function will try to read the data.

Result :

success – returns byte array containing the read data.
error – returns *null* or *Exception*.

Description :

Reads data from transponder memory with ID provided by *_tagID*. This function blocks the execution until data is successfully read, or for a maximum of time provided by parameter.

public async Task<bool> WriteBytesAsync(byte[] _tagID, int _page, int _from, byte[] _data, bool _lock)

Parameter :

_tagID – byte array representing the ID of the transponder to read memory from. The value to pass to this function can be obtained using *Identify* function. maximum time in milliseconds that the process will try to read the information from reader.
_page – memory page/bank to be read (only supported by some transponder types)
_from – first memory position to be read (being 0 the first byte of the selected memory)
_data – data to write to transponder memory.
_lock – if set to true, the memory blocks will be locked after writing (Attention: locking the blocks is irreversible and blockwise! See transponder chip documentation for more information)

Result :

success – returns *true* if data successfully written.
error – returns *false* or *Exception*.

Description :

Writes data into transponder memory with ID provided by *_tagID*. This function blocks the execution until data is successfully read, or for a maximum of 1000ms.

public async Task<bool> WriteBytesAsync(byte[] _tagID, int _page, int _from, byte[] _data, bool _lock, int _maxScanTimeMs)

Parameter :

_tagID – byte array representing the ID of the transponder to read memory from. The value to pass to this function can be obtained using *Identify* function. maximum time in milliseconds that the process will try to read the information from reader.
_page – memory page/bank to be read (only supported by some transponder types)
_from – first memory position to be read (being 0 the first byte of the selected memory)
_data – data to write to transponder memory.
_lock – if set to true, the memory blocks will be locked after writing (Attention: locking the blocks is irreversible and blockwise! See transponder chip documentation for more information)
_maxScanTimeMs – maximum time in milliseconds that the function will try to write the data.

Result :

success – returns *true* if data successfully written.
error – returns *null* or *Exception*.

Description :

Writes data into transponder memory with ID provided by *_tagID*. This function blocks the execution until data is successfully read, or for a maximum of time provided by parameter.

public async Task<TELIDSensorResultInfo> GetSensorDataAsync(int _sensorType)

Parameter :

_sensorType – this parameter selects the TELID® sensor type to search for.
Default values:
HF (1356): 0xFC – searches for all supported TELID®200 types
UHF (868): 0x00 – searches for all supported TELID®400 types

Result :

success – returns instance of *TELIDSensorInfo* representing the found TELID® and the measured value.
error – returns *null* or *Exception*.

Description :

Scans for a TELID® sensor transponder and performs a measurement. This function blocks the execution until data is successfully read, or for a maximum of 1000ms.

public async Task<TELIDSensorResultInfo> GetSensorDataAsync(int _sensorType, int _maxScanTimeMs)

Parameter :

_sensorType – this parameter selects the TELID® sensor type to search for.
Default values:
HF (1356): 0xFC – searches for all supported TELID®200 types
UHF (868): 0x00 – searches for all supported TELID®400 types
_maxScanTimeMs – maximum time in milliseconds that the function will try search for a TELID® sensor transponder.

Result :

success – returns instance of *TELIDSensorInfo* representing the found TELID® and the measured value.
error – returns *null* or *Exception*.

Description :

Scans for a TELID® sensor transponder and performs a measurement. This function blocks the execution until data is successfully read, or for a maximum of time provided by parameter.

RFID functions – running in background threads

The following chapter describes versions of functions above that will perform the operation in a background thread and report the result using *DocResultChanged* event. The functions are hardware independent and abstracted from different transponder systems based on system-specific functions described in the next chapters. Using these functions is more comfortable, but sometimes also time-expensive, so in time-critical processes the usage of system-specific functions is recommended.

These functions encapsulate reader, frequency, and tag-specific functions to abstract functions. If access to TAG-specific functions is required, you may need to use specific functions instead or additional to these.

public void StopThreadOperations()

Parameter :
-
Result :
-
Description :
Stops all running thread operations. This function completes immediately after being call, and event *DocResultChanged* is raised one last time to notify thread process is completed.

public void StartIdentify(int _repeatCount, int _delayBetweenSearchMs, int _notifySuccessOnly)

Parameter :
_repeatCount – number of times the thread will repeat the process. Providing 0 as value will cause the thread to repeat the process until it is stopped.
_delayBetweenSearchMs – time in milliseconds that the thread will wait before performing the next internal *Identify* operation.
_notifySuccessOnly – If true, the *DocResultChanged* event will be raised only if a transponder is found.
Result :
-
Description :
Starts a *Identify* process in a background thread. It scans for a TAG near antenna interface. This function completes immediately after being call, and results are provided using *DocResultChanged* event.

public void StartReadBytes(byte[] _tagID, int _page, int _from, int _length)

Parameter :

_tagID – byte array representing the ID of the transponder to read memory from. The value to pass to this function can be obtained using *Identify* function. maximum time in milliseconds that the process will try to read the information from reader.
_page – memory page/bank to be read (only supported by some transponder types)
_from – first memory position to be read (being 0 the first byte of the selected memory)
_length – number of bytes to be read.

Result :

-

Description :

Starts a *ReadBytes* process in a background thread. It reads data from transponder with ID provided by *_tagID*. This function completes immediately after being call, and results are provided using *DocResultChanged* event. The *ReadBytes* process will be performed until successful or a maximum of 5 times.

public void StartReadBytes(byte[] _tagID, int _page, int _from, int _length, int _maxTryCount)

Parameter :

_tagID – byte array representing the ID of the transponder to read memory from. The value to pass to this function can be obtained using *Identify* function. maximum time in milliseconds that the process will try to read the information from reader.
_page – memory page/bank to be read (only supported by some transponder types)
_from – first memory position to be read (being 0 the first byte of the selected memory)
_length – number of bytes to be read.
_maxTryCount – maximum number of times the *ReadBytes* process will be performed in case previous calls fail.

Result :

-

Description :

Starts a *ReadBytes* process in a background thread. It reads data from transponder with ID provided by *_tagID*. This function completes immediately after being call, and results are provided using *DocResultChanged* event. The *ReadBytes* process will be performed until successful, or a maximum of times defined by parameter.

public void StartWriteBytes(byte[] _tagID, int _page, int _from, byte[] _data)

Parameter :

_tagID – byte array representing the ID of the transponder to read memory from. The value to pass to this function can be obtained using *Identify* function. maximum time in milliseconds that the process will try to read the information from reader.
_page – memory page/bank to be read (only supported by some transponder types)
_from – first memory position to be read (being 0 the first byte of the selected memory)
_data – data to write to transponder memory.

Result :

-

Description :

Starts a *WriteBytes* process in a background thread. It writes data into transponder memory with ID provided by *_tagID*. This function completes immediately after being call, and results are provided using *DocResultChanged* event. The *WriteBytes* process will be performed until successful or a maximum of 5 times.

public void StartWriteBytes(byte[] _tagID, int _page, int _from, byte[] _data, int _maxTryCount)

Parameter :

_tagID – byte array representing the ID of the transponder to read memory from. The value to pass to this function can be obtained using *Identify* function. maximum time in milliseconds that the process will try to read the information from reader.
_page – memory page/bank to be read (only supported by some transponder types)
_from – first memory position to be read (being 0 the first byte of the selected memory)
_data – data to write to transponder memory.
_maxTryCount – maximum number of times the *WriteBytes* process will be performed in case previous calls fail.

Result :

-

Description :

Starts a *WriteBytes* process in a background thread. It writes data into transponder memory with ID provided by *_tagID*. This function completes immediately after being call, and results are provided using *DocResultChanged* event. The *WriteBytes* process will be performed until successful or a maximum of times defined by parameter.

public void StartGetSensorData(int _sensorType, int _repeatCount, int _delayBetweenSearchMs, bool _notifySuccessonly)

Parameter :

_sensorType – this parameter selects the TELID® sensor type to search for. Default values:

HF (1356): 0xFC – searches for all supported TELID®200 types

UHF (868): 0x00 – searches for all supported TELID®400 types

_repeatCount – number of times the thread will repeat the process. Providing 0 as value will cause the thread to repeat the process until it is stopped.

_delayBetweenSearchMs – time in milliseconds that the thread will wait before performing the next internal *GetSensorData* operation.

_notifySuccessOnly – If true, the *DocResultChanged* event will be raised only if a sensor transponder is found.

Result :

-

Description :

Starts a *GetSensorData* process in a background thread. It scans for a TELID® sensor transponder and performs a measurement. This function completes immediately after being call, and results are provided using *DocResultChanged* event.

Classes, Enumerations, Events and Delegates

DocInterfaceControl properties

This chapter includes all the available properties, which can be used to get the status of the initialization process and configure some parameters.

public bool IsInitialized

Accessors :
get / -

Description :
Returns the status of the communication interface. Result is *true* if the communication interface is still open, *false* otherwise.

public bool IsInitializing

Accessors :
get / -

Description :
Returns the status of the initialization process. Result is *true* if the initialization process is still running, *false* otherwise.

public ICommunicationHandler CommunicationHandler

Accessors :
get / -

Description :
Returns the internal instance of *ICommunicationHandler* used to communicate with the RFID reader. This property may be used to initialize a new instance of *DocInterfaceControl* that communicates using the same host-side peripheral interface.

public int InterfaceType

Accessors :
get / -

Description :
Returns the interface type used to initialize the *DocInterfaceControl* class.

public int ProtocolType

Accessors :
get / set

Description :
Access the selected protocol type currently being used in *DocInterfaceControl* class to communicate with the reader.

public uint SystemMask

Accessors :
get / set

Description :
Access the selected system mask currently being used in *DocInterfaceControl* class. This value consists on multiple flags that allow the configuration of the transponder systems supported by RFID operations described below. For possible values, see *SystemMaskEnum*

DocInterfaceControl delegates

This chapter describes delegates that define events used in this *DocInterfaceControl* class.

public delegate void InitializeCompletedEventHandler(object _sender, bool _portOpen)

Parameter :
_sender – object instance where the event handler is attached.
_portOpen – true if the communication port is open as result of *Initialize* function.
Result :
-
Description :
Represents a method that will handle the result to an Initialize process.

public delegate void DocResultEventHandler(object _sender, DocResultEventArgs _docResult)

Parameter :
_sender – object instance where the event handler is attached.
_docResult – instance of *DocResultEventArgs*
Result :
-
Description :
Represents a method that will handle the result to an RFID operation running in a separate background thread.

DocInterfaceControl events

This chapter describes events defined in *DocInterfaceControl* class.

public event InitializeCompletedEventHandler InitializeCompleted

Description :
This event is raised when Initialize process is completed after calling *StartInitialize* function.

public event DocResultEventHandler DocResultChanged

Description :
This event is raised when any RFID operation running in separate background thread has a result to report.

Utils classes

This chapter describes classes used as return types of functions described above.

iIDReaderLibrary.Utils.ReaderIDInfo

This class contains information about the Reader.

Type	Property name	Description
int	ReaderID	Reader-ID (Serial number) of the RFID reader
string	HwInfo	String representation of the reader Hardware version
string	FwInfo	String representation of the running Firmware
string	HexString	Hexadecimal representation of class contents

iIDReaderLibrary.Utils.HfScanResultInfo

This class contains result information about a HF scan.

Type	Property name	Description
byte[]	TagID	Transponder ID bytes
string	TagID_Hex	Hexadecimal representation of the Transponder ID

iIDReaderLibrary.Utils.UhfScanResultInfo

This class contains result information about a UHF scan.

Type	Property name	Description
List<UHF_TagScanInfo>	TagInfoList	List of UHF TAGs

iIDReaderLibrary.Utils.TELIDSensorResultInfo

This class contains result information about a TELID® sensor transponder and the measured sensor value.

Type	Property name	Description
int	SerialNumber	Serial number
string	SerialNumberHex	Hexadecimal representation of the serial number
string	Description	Description containing the type of sensor
SensorMeasurement[]	Measurements	Array of sensor measurements read through TELID® sensor transponder

iIDReaderLibrary.Utils.UHF_TagScanInfo

This class contains result information about an individual UHF transponder.

Type	Property name	Description
UHF_TagUII	UII	UII information of transponder
int	AntennaNumber	Antenna number where the transponder was found
double	RSSI	RSSI value if available, *NaN* otherwise

iIDReaderLibrary.Utils.UHF_TagUII

This class contains UII information of a UHF transponder.

Type	Property name	Description
byte[]	PC	PC bytes
byte[]	XPC_W1	XPC_W1 bytes (if available, depending on UII configuration)
byte[]	XPC_W2	XPC_W2 bytes (if available, depending on UII configuration)
byte[]	UII	UII bytes, representing the Transponder ID
string	UII_Hex	Hexadecimal representation of the UII
bool	IsDIN15459UID	True if UII bytes represent a DIN15459 UID (if T bit in PC byte is 1)
bool	IsEPC	True if UII bytes represent an EPC (if T bit in PC byte is 0)
bool	IsQC_UID	True if UII bytes represent a DIN15459 UID with “QC” as Issuing Agency
bool	IsMssUID	True if UII bytes represent a UID assigned by microsensys
byte	AFI	Obtain the AFI if the UII bytes represent an DIN15459 UID

iIDReaderLibrary.Utils.ReadBytesResultInfo

This class contains result information about data read from a transponder memory.

Type	Property name	Description
byte[]	TagID	Transponder ID where data was read from
int	FromPage	Memory page/bank
int	FromByte	First byte position read
byte[]	ReadResult	Bytes read from transponder memory

iIDReaderLibrary.Utils.WriteBytesResultInfo

This class contains result information about data written into a transponder memory.

Type	Property name	Description
byte[]	TagID	Transponder ID where data was read from
int	FromPage	Memory page/bank
int	FromByte	First byte position read
bool	WriteResult	True if data successfully written
byte[]	WrittenBytes	Bytes written into transponder memory

iIDReaderLibrary.Utils.SensorMeasurement

This class contains sensor measurement details.

Type	Property name	Description
DateTime	Timestamp	Timestamp when the measurement took place
SensorValue[]	Values	Array of measured sensor values

iIDReaderLibrary.Utils.SensorValue

This class contains individual sensor value details.

Type	Property name	Description
double	Magnitude	Magnitude of measured sensor value
string	Symbol	Symbol representing the sensor value type
string	Unit	Unit representing the sensor value unit
SensorPhysicalSizes	ValueType	Enum value representing the sensor value type

iIDReaderLibrary.Utils.DocResultEventArgs

This class contains result information of a RFID operation running in background thread.

Type	Property name	Description
DateTime	Timestamp	Timestamp when result is initialized
bool	ProcessFinished	If true, the operation running in background thread finished
object	ResultInfo	Object representing the result of the background process. Object may be *null* when the operation has no data to report
Exception	ResultException	Instance of Exception representing the Exception caused in background thread

ResultInfo may contain different object types depending on the running operation:

StartIdentify :
HF (1356): Object of type *HfScanResultInfo*
UHF (868): Object of type *UhfScanResultInfo*

StartReadBytes :
Object of type *ReadBytesResultInfo*

StartWriteBytes :
Object of type *WriteBytesResultInfo*

StartGetSensorData :
Object of type *TELIDSensorResultInfo*

iIDReaderLibrary.Utils.InterfaceCommunicationSettings

This class provides a series of functions needed to initialize the interface communication.

ICommunicationHandler InitCommHandler()

Parameter :
-

Result :
Instance of class that implements *ICommunicationHandler* interface.

Description :
Initializes a new instance of a class that implements *ICommunicationHandler* and returns it as a result.

static string[] GetAvailablePortNames()

Parameter :
-

Result :
Array of string containing port names.

Description :
Reads the serial port names available in host device and populates array of string. These names can be used to initialize a new *DocInterfaceControl* instance.

static string[] GetAvailableBlePairedDevices()

Parameter :
-

Result :
Array of string containing device names.

Description :
Reads the paired BluetoothLE device names available in host device and populates array of string. These names can be used to initialize a new *SpcInterfaceControl* instance.

static InterfaceCommunicationSettings GetForUsbDevice()

Parameter :
-

Result :
Instance of *InterfaceCommunicationSettings* class for USB communication.

Description :
Initializes a new instance of *InterfaceCommunicationSettings* with default baud rate of 57600 to pass to *DocInterfaceControl* constructor.

static InterfaceCommunicationSettings GetForUsbDevice(int _baudrate)

Parameter :
_baudrate – baud rate value

Result :
Instance of *InterfaceCommunicationSettings* class for USB communication using the given parameters.

Description :
Initializes a new instance of *InterfaceCommunicationSettings* to pass to *DocInterfaceControl* constructor.

static InterfaceCommunicationSettings GetForSerialDevice(int _portType, string _portName)

Parameter :
_portType – number representing the port type. See *PortTypeEnum*
_portName – name of the serial port (not needed for USB port type)

Result :
Instance of *InterfaceCommunicationSettings* class representing the given parameters.

Description :
Initializes a new instance of *InterfaceCommunicationSettings* with default baud rate of 57600 to pass to *DocInterfaceControl* constructor.

static InterfaceCommunicationSettings GetForSerialDevice(int _portType, string _portName, int _baudrate)

Parameter :
_portType – number representing the port type. See *PortTypeEnum*
_portName – name of the serial port (not needed for USB port type)
_baudrate – baud rate value

Result :
Instance of *InterfaceCommunicationSettings* class representing the given parameters.

Description :
Initializes a new instance of *InterfaceCommunicationSettings* to pass to *DocInterfaceControl* constructor.

static InterfaceCommunicationSettings GetForPcanDevice(int _portType, string _portName, uint _pcanLocalAddr, uint _pcanRemoteAddr)

Parameter :

_portType – number representing the port type. See *PortTypeEnum*
_portName – name of the serial port (not needed for USB port type)
_pcanLocalAddr – PCAN address to use for host device.
_pcanRemoteAddr – PCAN address of the reader.

Result :

Instance of *InterfaceCommunicationSettings* class representing the given parameters.

Description :

Initializes a new instance of *InterfaceCommunicationSettings* to pass to *DocInterfaceControl* constructor.

static InterfaceCommunicationSettings GetForLanDevice(int _portType, IPEndPoint _localEndPoint, IPEndPoint _remoteEndPoint, uint _pcanLocalAddr, uint _pcanRemoteAddr)

Parameter :

_portType – number representing the port type. See *PortTypeEnum*
_localEndPoint – instance of System.Net.IPEndPoint representing the host device IP and port.
_remoteEndPoint – instance of System.Net.IPEndPoint representing the reader IP and port number.
_pcanLocalAddr – PCAN address to use for host device.
_pcanRemoteAddr – PCAN address of the reader.

Result :

Instance of *InterfaceCommunicationSettings* class representing the given parameters.

Description :

Initializes a new instance of *InterfaceCommunicationSettings* to pass to *DocInterfaceControl* constructor.

Enumerations

This chapter describes enumerations that can be used in various functions described above.

iIDReaderLibrary.Utils.Definitions.InterfaceTypeEnum

This enumeration contains values for the supported interface type values.

Name	Value	Description
Interface_HF	1356	Used for HF communication (13.56MHz)
Interface_UHF	868	Used for UHF communication

iIDReaderLibrary.Utils.Definitions.PortTypeEnum

This enumeration contains values for the supported protocol type values.

Name	Value	Description
PortType_Serial	0	RS232 or USB in VCP mode
PortType_Bluetooth	2	Bluetooth™ SPP
PortType_USB	4	USB interface (no VCP mode)
PortType_BluetoothLE	5	Bluetooth™ Low Energy SPP

iIDReaderLibrary.Utils.Definitions.ProtocolTypeEnum

This enumeration contains values for the supported protocol type values.

Name	Value	Description
ProtocolType_2000	2	iID® 2000 protocol
ProtocolType_3000	3	iID® 3000 protocol
ProtocolType_v4	4	iID® interface protocol V4
ProtocolType_LEGIC	0x1002	LEGIC™ direct protocol

iIDReaderLibrary.Utils.Definitions.SystemMaskEnum

This enumeration contains values for the supported system mask values.

Name	Value	Description
GROUP_ISO15693	0x01	13.56MHz ISO15693
GROUP_IID_L	0x02	13.56MHz IID®-L
GROUP_IID_D	0x04	13.56MHz IID®-D
GROUP_IID_G	0x08	13.56MHz IID®-G
GROUP_64BITRO	0x10	13.56MHz IID®-N
GROUP_ICODEUID	0x20	13.56MHz I-CODE-UID
GROUP_LEGIC_UID	0x40	13.56MHz LEGIC-PRIME
GROUP_ICODE1	0x80	13.56MHz I-CODE-1
GROUP_ISO14443A	0x100	13.56MHz ISO14443-A
GROUP_PICOTAG	0x200	13.56MHz PICO-TAG
GROUP_IIDP	0x400	13.56MHz IID®-P
GROUP_ISO14443B	0x800	13.56MHz ISO14443-B
GROUP_ISO15693_TRANSP	0x10000000	13.56MHz ISO15693 addressed transparent (to be used with GROUP_ISO15693)
GROUP_ISO15693_RWblocks	0x20000000	13.56MHz ISO15693 RWblocks (to be used together with GROUP_ISO15693)
GROUP_LEGIC_FS	0x40000000	13.56MHz LEGIC file system (to be used together with ISO type above)
GROUP_DUALECHO	0x80000000	13.56MHz DUALECHO (not compatible with GROUP_ISO14443-A)

iIDReaderLibrary.Utils.Definitions.TagTypesEnum

This enumeration contains values for the supported chip types.

Name	Value	Description
ICODESLI	0x04	ISO15693 I-Code SLI
TAGIT	0x07	ISO15693 tag-it
EM4135	0x16	ISO15693 iID®-M
MYD_ONLY	0x05	ISO15693 my-D custom mode
ICODEUID	0x41	ISO15693 I-Code UID
ECHO_DUAL_HF_UHF	0x90	EM echo
ST25_ISO15693	0xA0	ISO15693 iID-M2
NTAG_203	0xB0	ISO14443-A NXP N-TAG 203
NTAG_213	0xB1	ISO14443-A NXP N-TAG 213
NTAG_215	0xB2	ISO14443-A NXP N-TAG 215
NTAG_216	0xB3	ISO14443-A NXP N-TAG 216
NTAG_I2C_1K	0xB4	ISO14443-A NXP N-TAG I²C 1K
NTAG_I2C_2K	0xB5	ISO14443-A NXP N-TAG I²C 2K
NTAG_210	0xB6	ISO14443-A NXP N-TAG 210
NTAG_212	0xB7	ISO14443-A NXP N-TAG 212
NTAG_213F	0xB8	ISO14443-A NXP N-TAG 213F
NTAG_216F	0xB9	ISO14443-A NXP N-TAG 216F
MIFARE_UL	0xC0	ISO14443-A Mifare Ultralight
MIFARE_UL_C	0xC1	ISO14443-A Mifare Ultralight C (not used)
MIFARE_CLASSIC_1K	0xC4	ISO14443-A Mifare Classic 1K
MIFARE_CLASSIC_4K	0xC5	ISO14443-A Mifare Classic 4K
MIFARE_MINI	0xC6	ISO14443-A Mifare Mini
MIFARE_PLUS_2K	0xC7	ISO14443-A Mifare Plus 2K
MIFARE_PLUS_4K	0xC8	ISO14443-A Mifare Plus 4K
MIFARE_DESFIRE_2K	0xC9	ISO14443-A Mifare Desfire 2K
ICODESLI_S	0xD4	ISO15693 I-Code SLI-S
ICODESLI_L	0xD5	ISO15693 I-Code SLI-L
ICODESLI_XS	0xD6	ISO15693 I-Code SLI-XS
ICODESLI_XL	0xD7	ISO15693 I-Code SLI-XL
ICODESLI_X	0xD8	ISO15693 I-Code SLI-X
ICODESLI_X2	0xD9	ISO15693 I-Code SLI-X2
IID2000	0xF0	iID-D
IC1	0xF1	I-Code 1
ARIO	0xF2	iID-N
IIDG	0xF3	iID-G
LEGIC	0xF4	LEGIC Prime
MYD_ISO	0xF5	ISO15693 my-D ISO mode
IIDL	0xF6	iID-L
PICOTAG_16K	0xF7	Pico-TAG 16k
IIDH	0xF8	iID-H
PICOTAG_32K	0xF9	Pico-TAG 32k
ISO14443B	0xFA	ISO14443-B iID-K
IIDQ1	0xFB	iID-Q1
IIDQ2	0xFC	iID-Q2
IIDP	0xFD	iID-P

Appendix

Error - Codetable

State	Error
0x00	no error
RFID interface error codes	
0x23	TAG-error
0x24	no TAG near antenna
0x26	OP-CODE unknown
0x28	protocol failure
0x29	unknown TAG instruction
0x2A	unknown TAG-error
0x2B	error writing to TAG
0x2C	error reading from TAG
0x2E	error control-reading TAG
0x2F	wrong data control-reading TAG (RAW)
0x30	error in CRC-CheckSum
others	Other hardware specific error codes may occur – see hardware documentation.
Driver error codes	
0x08	identifiers do not match
0x11	range error
0x14	General TAG-error
0x3F	Communication or port error
0x43	Configuration error
0x4F	unknown/not supported option detected
0xFF	general communication or driver error

See hardware documentation for further error codes.