API Documentation

(short form description)

Product / Project:	**microsensys iID® 3000 Java API for Windows**
Customer / Project Code:	**-**
Product:	**MICROSENSYS iID®3000 PRO RFID interfaces**
Product Code:	**-**
Document Revision:	**6.6 preli**
Date:	**2021-09-30**

API - Definition for Windows devices

This document describes the java class library for Windows devices supporting iID® 3000 PRO, iID® v4 and LEGIC® interfaces.

Driver name:	microsensys.RFIDFunctions
Tested OS:	Windows 10 64bit, Ubuntu 64bit
Supported host interface:	Bluetooth(TM) SPP, Serial Communication
Revision:	----
Package:	microsensys.RFIDFunctions
Class:	RFIDFunctions

API Documentation

(short form description)

1. Short history

This chapter includes a short history of modifications.

Date	Reason	Modification	Release Date / Version	FileName
2011-10	- base document creation		-	microsensys.RFIDReader
2018-09	- first official version Windows interface		2018-09 / 5.8	
2019-02	- bugfix read/write ISO14443A - added support for new TELID®2xx - Improved Tag identification for ISO14443A	(Added note for "getTagMaxLength")	2019-01 / 5.10	
2020-07	- implementation of USB and BLE communication - added function for LEGIC® personal badge - added some special ISO15693, ISO14443 functionality - added "custom user memory" functions - solved Baudrate bug	New port type and function (only for RFIDFunctions_LEGIC)	2020-07 / 6.2	
2021-06	- added support for power states - "readReaderID()" now reads firmware build info (if available) - added support for TELID®200.m		2021-06 / 6.4	
2021-09	no changes		2021-09 / 6.6	

2. *Bluetooth™ and serial port connection functions*

public RFIDFunctions(int portType) [constructor]

Initializes class. PortType possible values:

Value	Description
0	- Serial communication (PortTypeEnum.Serial)
2	- Bluetooth communication (PortTypeEnum.Bluetooth)

public void initialize()

Function connects to RFID device (open serial port, create Bluetooth connection). The connection procedure could take a while depending on devices. This function does not block execution. During this procedure, the function "isConnecting" will return "true".

public void terminate()

Function disconnects from device (close serial port, close Bluetooth connection).

public boolean isConnected()

Function checks if the connection to RFID device is still open.

public boolean isConnecting()

Function checks if the connecting procedure is still running.

public void setPortName(String PortName)

Function sets the serial Port Name to which the device is connected. This parameter should be set before using "initialize".

public void setTimeout(int timeout)

Function sets the driver timeout value.

public int getTimeout()

Function gets the driver timeout value.

public void setProtocolType(int protocolType)

Function sets the protocol to be used in the communication with the Reader. Possible values:

Value	Description
3	- iID® 3000 protocol (supports HF) [default] (ProtocolTypeEnum.Protocol_3000)
4	- iID® v4 protocol (supports UHF) (ProtocolTypeEnum.Protocol_v4)
4098	- LEGIC® protocol (supports LEGIC®) (ProtocolTypeEnum.Protocol_LEGIC

public int getProtocolType()

Function gets the currently selected protocol.

public void setInterfaceType(int interfaceType)

Function sets the interface type (frequency) used by the Reader. Possible values:

Value	Description
1356	- HF RFID reader (InterfaceTypeEnum.HF)
868	- UHF RFID reader (InterfaceTypeEnum.UHF)

public int getInterfaceType()

Function gets the currently selected interface type.

public void setPage(int pageNum)

Function sets the transponder page/segment number to be read/written when readBytes/writeBytes is called.

public int getPage()

Function gets the currently selected transponder page/segment.

public long getSystemMask()

Function to get the system mask (what kind of tags will be able to read). The return value is the system mask value. (See iID® driver documentation)

public void setSystemMask(long systemMask)

Function to set the system mask (what kind of tags will be able to read). (See iID® driver documentation)

3. Reader functions

public ReaderIDInfo readReaderID()

Function to read the ID of the RFID Reader (it turns off the reader antennae if it was on). The return value is an instance of the class ReaderIDInfo (see point 5 in this document: Utils) or null in case of an error.

public ReaderModeInfo getReaderMode() throws *MssException*

Function to get the current Reader mode information. The return value is an instance of ReaderModeInfo (see point 5 in this document: Utils). This function throws an *MssException* when an error occurs during execution.

public boolean setReaderMode(ReaderModesEnum _mode) throws *MssException*

Function to set the RFID Reader into a determinate mode (Only for SPC-compatible Readers). The return value is true If the bytes were successfully written or false otherwise. This function throws an *MssException* when an error occurs during execution. There are currently only two modes available:

ReaderModesEnum	Description
DOC	- Selecting this option, (if the Reader is in SPC mode) the Script will stop and the DOC (Direct Online Communication) mode established.
SPC	- Selecting this option, the Reader will be set in SPC mode. This procedure is completed after a device restart. **Warning: If the appropriate Script is not loaded into the device at the time of calling this function, the Reader could land in an indeterminate state and a Factory-Reset will be needed! It is recommended to check if the script is loaded using "getReaderMode" function. To load the script, please see our QuickStart Guide.**

public byte[] readCustomUserMemory() throws MssException

Function to read the custom user initialization written in the reserved reader memory. This function throws an *MssException* when an error occurs during execution.

public boolean programCustomUserMemory(byte[] _customUserContents) throws MssException

Function to program a custom initialization in the reserved reader memory. Maximum allowed length is 63 Bytes. This function throws an *MssException* when an error occurs during execution.

4. RFID functions

public byte[] identify() throws *MssException*

Function search for a TAG and get its UID. The return value is either the UID coded in a byte array or null if there was no transponder near the antenna. This function throws an *MssException* when an error occurs during execution.

public byte[] readBytes(byte[] ID, int from, int length) throws *MssException*

Function to write data into a transponder device. The return value is either the data to be read or null if there was no transponder near the antenna. This function throws an *MssException* when an error occurs during execution.

Parameters:

Name	Description
byte[] ID	- ID of the TAG to be read.
int from	- Address of first byte of the TAG that will be read.
int length	- Number of bytes to read.

public boolean writeBytes(byte[] ID, int from, byte[] data, boolean lock) throws *MssException*

Function to write data into a transponder device. The return value is true if the bytes were successfully written or false otherwise. This function throws an *MssException* when an error occurs during execution.

Parameters:

Name	Description
byte[] ID	- ID of the TAG which to be written.
int from	- Address of first byte of the TAG that will be written.
byte[] data	- Bytes to write in the TAG
boolean lock	- Determines if the block will be locked or not (ISO15693)

public String getTagParameters()

Function to get some TAG parameters. The function performs a search operation and returns the found TAG parameters. The return value is a String with the info of the found TAG. If there was an error, the String is "ERROR", if not; it contains the manufacturer, tag type, and memory length.

public int getTagMaxLength()

Function to get the max length of the memory of the tag last found. The return value is an int with this value. If the TAG is unknown or the identify function was not used before, the value is 0.

NOTE: This function uses the last read UID to calculate the value. For handling ISO14443A transponders, a previous call to "getTagParameters" function is needed in order to calculate TAG memory size.

public TELIDSensorInfo getSensorData(int sensorType) throws *MssException*

Function to get a value from a TELID® sensor. The function performs a search operation and gets its sensor data. The return value is an instance of TELIDSensorInfo (see point 5 in this document: Utils) or null when no sensor was found. The variable "sensorType" is used to choose between the different sensor TAGs. This function throws an *MssException* when an error occurs during execution.

Name	Description
int sensorType	- Used to select between the different TELID® sensors. ○ UHF default value is 0 (auto switch between types) ○ HF default value is 0xFC (auto detect)

public byte[] receiveOutputData() throws *MssException*

Function to get data sent by the RFID Reader in Script-Mode. This data needs to be sent using "Host Output" command and "3000prot" or "v4 prot" as Format. The return value is the data received, which should be the raw data sent by the reader (the function removes the selected Format). This function throws an *MssException* when an error occurs during execution.

public byte[] receiveOutputData(boolean _clearBuffer) throws *MssException*

Similar to "receiveOutputData()" but with the option to decide if the receive buffer should be cleared before receiving or not. This function throws an *MssException* when an error occurs during execution.

public boolean sendScriptData(byte[] _scriptData, boolean _useProtocol) throws *MssException*

Function to send data to the RFID Reader in Script-Mode. The data to send should be given using "_scriptData" parameter. If "_useProtocol" is true, the data is sent using v4 Protocol. If false, the data will be sent only with an "~" prepended. This data will be received by "GetTrigger" function in Script. The return value is true if the data was successfully sent. This function throws an *MssException* when an error occurs during execution.

public String readPersonalBadgeID(byte[] _UID) throws *MssException*

Function to read the Personal Badge ID number (LEGIC® card). The UID of the found card can be given as parameter. The function will then read only the badge ID from the specified card. If the parameter "_UID" is null, the function will search for a transponder and read the Personal Badge ID. The return value is the text interpretation of the badge. This function throws an *MssException* when an error occurs during execution.

5. Utils

ReaderIDInfo

This class contains the information of the RFID reader and implements some functions to provide it to the developer/user:

Function Name	Description
int getReaderID()	- Returns the Reader ID number (Serial Number).
byte[] getBytes()	- Returns the bytes received from the Reader.
String toHexString()	- Returns the string representation as HEX string of the received bytes.
String toString()	- Returns a text representation of the information contained in the class (provided by the reader)

TELIDSensorInfo

This class contains the information of the currently found TELID® and implements some functions to provide it to the developer/user:

Function Name	Description
int getSerialNumber()	- Returns the TELID® Serial Number (implemented in our 2xx series only).
byte[] getID_Bytes()	- Returns the bytes representing the ID of the TELID®.
String getTELIDDescription()	- Returns the string representation of the product, including product number and sensor type.
double[] getSensorValues()	- Returns an array with the sensor values read by the reader.
String[] getSensorUnits()	- Returns an array with the corresponding units of each measurement.
String[] getSensorValueStrings()	- Returns an array with the string representation of each value read by the reader.
String toString()	- Returns a text representation of the information contained in the class (provided by the reader).

ReaderModeInfo

This class contains the information about the RFID Reader mode and implements some functions to provide it to the developer/user:

Function Name	Description
ReaderModesEnum getMode()	- Returns a "ReaderModesEnum" (de.microsensys.utils) value representing the Reader mode
String getScriptName()	- Returns the script representing the script name contained in the Reader-memory
String toString()	- Returns a text representation of the information contained in the class (provided by the reader).

InterfaceTypeEnum

This class contains the different values possible for the "InterfaceType" parameter. This value is used for "setInterfaceType(int)" function. Possible values:

Name	Description
LF	- Value for LF reader (currently not supported)
HF	- Value for HF reader
UHF	- Value for UHF reader

PortTypeEnum

This class contains the different values possible for the "PortType" parameter. This value is in RFIDFunctions constructor. Possible values:

Name	Description
Serial	- Value for serial interface
Bluetooth	- Value for Bluetooth interface, to use an external Bluetooth reader paired with the device using the Virtual COM-Port

ProtocolTypeEnum

This class contains the different values possible for the "ProtocolType" parameter. This value is used for "setProtocolType(int)" function. Possible values:

Name	Description
Protocol_3000	- Value for 3000pro reader protocol (normally HF readers)
Protocol_v4	- Value for v4pro reader protocol (normally UHF readers)
Protocol_LEGIC	- Value for LEGIC® reader protocol (only HF LEGIC® readers)

API Documentation

(short form description)

Erfurt, 2021-09-30

This document is preliminary and subject to change.