TECHNICAL INFORMATION MANUAL

Revision 1 – 07/12/2023

CAEN RFID SDK

Application Program Interfaces for:

Visual C

.NET Standard 2.0

Swift

Java

Light C

Scope of the Manual

This manual documents the API used by C, Java, Android, .Net and Swift programmers who want to write applications for controlling and using CAEN RFID readers.

Change Document Record

Date	Revision	Changes	Pages
07 Dec 2023	01	First release	-

Reference Document

[RD1] EPCglobal: EPC Radio-Frequency Identity Protocols Class-1 Generation-2 UHF RFID Protocol for Communications at 860 MHz – 960 MHz, Version 2.0.1 (April 2015).

CAEN RFID srl

Via Vetraia, 11 55049 Viareggio (LU) - ITALY
Tel. +39.0584.388.398 Fax +39.0584.388.959
info@caenrfid.com
www.caenrfid.com

Index

List of Tables

1 INTRODUCTION

Overview on SDK

CAEN RFID provides a Software Development Kit (SDK) aimed to facilitate the software developers in interfacing with its readers. The SDK provides Application Program Interfaces (API) for the following programming languages: Visual C, .NET Standard 2.0, Swift, Java (one API for general pc and one for Android) and C for microcontrollers (Light C).

The functionalities and the behaviors exported by the libraries are exactly the same for all the languages but, due to different syntax and target environments, there are differences in the implementation of functions and methods. Implementations for .NET Standard, Java (PC/Android) and Swift are very similar because they are both Object Oriented environments while the Visual C and Light C differs more.

The Object Oriented APIs implementation (Java and .NET) defines a set of classes that models the devices characteristics, the main one are the CAENRFIDReader class and the CAENRFIDLogicalSource class. The first one implements the main methods used to configure general readers' parameters like the output power, the airlink protocol, the internal battery level, GPIO handling and so on, the latter provides the methods to be used in order to communicate with the RFID tags (tags detection, read and write commands and so on).

The C (Visual C and Light C) implementation, on the contrary, is an imperative language so it implements a set of data types (defined into the CAENRFIDTypes.h header file) and a list of functions (defined into the CAENRFIDLib.h header file) in order to obtain the same functionalities Object Oriented Languages.

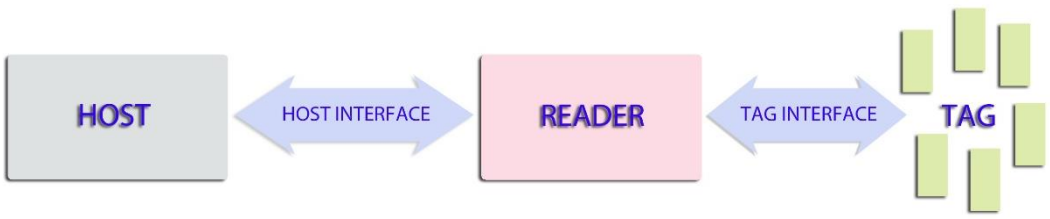
In the Object Oriented languages there are some methods that return objects, these methods have no correspondent in C language.

The following paragraphs will denote the differences in functionality for the topics listed below:

- Functions and methods names
- Error Handling
- Managing connections with the readers
- Return data mechanism
- Passing parameters to methods and functions

SDK Design Concepts

A CAEN RFID reader can be seen, in a simplified model, as a box with one or more communication interfaces on the host's side and one or more antennas on the tags' side. The reader accepts commands coming from an host (a PC or any other controlling device), it uses the interface to the tag (typically one or more antennas) to perform operations on the tags and it replies to the host.



The API defines a number of classes (emulated in C language by functions and data types) in order to represent this simplified model; two of them define most of the methods and can be considered the core classes of the API itself: *CAENRFIDReader* and *CAENRFIDLogicalSource*.

CAENRFIDReader class provides methods for the general reader configuration, host communication interfaces configuration, HW parameters etc.

CAENRFIDLogicalSource class defines the methods for the reader to tag communication and its configuration.

Working with the host interfaces, mostly represented in the *CAENRFIDReader* class, is quite straightforward since they are standard communication interfaces (Ethernet, RS232 and similar) while for the *CAENRFIDLogicalSource* class a deeper insight is needed.

One or more physical RFID antennas (called ReadPoints in the API) can be connected to a CAEN RFID, each one able to detect tags and it is typical, in UHF RFID installations, to place multiple antennas in the same place for a better coverage of the reading area. In this case, even if multiple antennas are used, the reading area is the same so, from a logical point of view, it is a single source of homogeneous information. In order to model this concept, we introduced the Logical Source concept that is implemented into the CAEN RFID API with the *CAENRFIDLogicalSource* class: it permits to group together ReadPoints (antennas) that are logically related. Each data exchange concerning tags is implemented through the CAENRFIDLogicalSource class, the case of a single antenna is a special case where a Logical Source contains a single Read Point.

The CAEN RFID API permits to add and remove Read Points to/from the Logical Source so that the user can easily represent its installation.

Up to the current revision, the API handles four Logical Source (called “Source_0”, “Source_1”, “Source_2”, “Source_3”) and four Read Points (called “Ant0”, “Ant1”, “Ant2”, “Ant3”) and the default configuration is that each LogicalSource contains only one different ReadPoint; in the future this could change and the number of Logical Sources and Read Points could be different depending on the model of the RFID reader.

On readers with a single antenna connector, the only meaningful Logical Source is the “Source_0” one and it contains the only one available antenna “Ant0”.

Note that after a reader switch off the Logical Sources composition is reset to the default configuration.

Functions and methods names

The functions and methods with the same functionalities have the same name in all languages. The only exceptions are due to the absence of the overloading feature in the C language: methods that are overloaded in Object Oriented API’s Languages are translated in a corresponding set of different functions in C.

Note: some methods and functions have changed name in the last revision of the API but older names are still functional to preserve backward compatibility (see § *Methods Description* page 34).

Error Handling

Object Oriented API’s languages handle error conditions using the exceptions mechanism: when a method encounters an error, an exception is thrown to the calling code. The API defines a proper class for the exception generated by its methods (CAENRFIDException) the origin of the error is represented inside the CAENRFIDException object as a string.

C language does not provide the exception mechanism, so the errors are handled using the return value of the functions. Each C function returns a numeric error code that can be interpreted using the CAENRFIDErrorCodes enumeration. Since no exceptions are generated, the execution flow of the program is not interrupted by the errors, so it is always suggested to check for error conditions in the code before to call other functions.

Managing connections with the readers

Object Oriented API’s languages allow to initiate and terminate the communication with the reader by means of two specific methods of the CAENRFIDReader objects. So, after an object of the class CAENRFIDReader is instantiated, the Connect method permits to start the communication with a reader while the Disconnect method permits to terminate the communication.

C language is not object oriented and the handling of the communication state is implemented using two functions. CAENRFID_Init is used to start the communication with a reader and to initialize all the library's internal data structures needed in order to maintain the communication active. The function returns a "handle" (very similar to the handles used in managing files) that have to be used in any subsequent function calls relative to that reader. At the end of the operation, a call to the CAENRFID_End function permits to close the communication link and to free the internal data structures.

Return data mechanism

As seen in the Error Handling paragraph, all the C functions return a numeric error codes. Due to that reason, functions that need to return data to the caller use output parameters. Output parameters for the C functions are highlighted in this manual by the underlined name in the formal parameter list.

Object Oriented API's languages use exceptions for the error handling so, typically, the data is returned to the caller using the return value of the methods.

Passing parameters to methods and functions

There are differences in the parameters' lists between Object Oriented languages methods and C functions. Many of those differences are due to the implicit reference of the methods to their objects. This characteristic of object-oriented languages is emulated in C functions using an additional explicit parameter. Methods belonging to CAENRFIDLogicalSource objects, for example, are emulated in C functions that accept SourceName parameters.

Other differences are due to the better handling of complex data types in Object Oriented languages. Arrays, for example, have implicit size that permit to pass a single parameter to methods requiring this data type. In C functions, passing an array as a parameter, need to specify both the memory address of the array and its size explicitly.

Note on Light C (C for microcontrollers)

API in Light C is not documented in this manual, due to the sliglhty syntax differences from other API languages, even with Visual C, and therefore the documentation is only available in digital format inside the SDK distributable package. All functions and data structures are commented inside the headers file of the library itself.

Note on Swift

Swift API has similar syntax to the other object-oriented languages based API, but there are some cases where syntax and structure of classes are so much different from other API that it's required a distinct section for Swift API itself.

Swift classes that differ to much from the other o.o.l. APIs will be documented with a dedicated paragraph next to the analogue class.

NOTE Although most of the features present in the other APIs are also present in Swift API, some features have not been implemented due to their very low rate usage or because are outdated.

2 GETTING STARTED WITH API

The minimum steps a programmer should follow in order to operate with a CAEN RFID reader using the API are the following:

- Open a connection with the reader
- Configuration of the logical source (optional if the default configuration is fine)
- Detection of tags and other operations on the tags
- Close the connection with the reader.

Here below a simple but complete code snippet showing the minimum required lines of code to detect RFID tags using a CAEN RFID reader and the CAEN RFID API. The code is shown using .NET C# programming language, but it can be easily adapted to the other languages supported by the API.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using com.caen.RFIDLibrary;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            CAENRFIDReader MyReader = new CAENRFIDReader();

            MyReader.Connect(CAENRFIDPort.CAENRFID_RS232, "COM3");

            CAENRFIDLogicalSource MySource = MyReader.GetSource("Source_0");

            CAENRFIDTag[] MyTags = MySource.InventoryTag();

            if (MyTags.Length > 0)
            {
                for (int i = 0; i < MyTags.Length; i++)
                {
                    String s = BitConverter.ToString(MyTags[i].GetId());
                    Console.WriteLine(s);
                }
            }
            Console.WriteLine("Press a key to end the program.");
            Console.ReadKey();
            MyReader.Disconnect();
        }
    }
}
```

3 BASIC OPERATIONS

Importing the libraries

All APIs are available as standalone libraries. To import them in the user application please refer your IDE (Integrated Development Environment) documentation.

The current version of .Net and Visual C libraries have been developed using Microsoft Visual Studio 2005 and the Java library has been developed using Oracle Netbeans 6.8.

Reader connection/disconnection

The first operation to perform in order to start to operate with CAEN RFID readers is to establish a connection. The connection method depends on the physical interface available on the readers that, at now, could be an Ethernet or a serial interface. Readers with USB interface can be considered as having a standard RS232 serial interface since the readers implement a USB to RS232 converter internally.

The API provides the *Connect* method/function that permits to establish the connection. The method accepts two parameters: the interface type and the address. If the interface type is serial the address will be the name of the serial port (e.g. "COM1"); if the interface is Ethernet (TCP/IP) the address will be the IP address of the reader.

Here below code examples for serial port connection and for TCP/IP connection are shown.

```
MyReader.Connect(CAENRFIDPort.RS232,"COM1");
MyReader.Connect(CAENRFIDPort.TCP,"192.168.0.2");
```

Once connected it is possible to operate on the reader with the other methods. At the end of the operations it is possible to disconnect from the reader using the *Disconnect* method/function.

```
MyReader.Disconnect();
```

Connect and *Disconnect* methods are members of the *CAENRFIDReader* class.

Getting information about the connected reader

The same API can be used to control different type of readers with different capabilities. It is often useful to know which type of reader is connected to the application in order to access to the right features.

The API provides two methods for getting information about the connected reader: *GetReaderInfo* and *GetFWRelease*. The first returns information about the reader model and serial number, the latter returns the version of the firmware running into the reader itself.

Here below code examples for getting information about the connected reader are shown.

```
CAENRFIDReaderInfo Info = MyReader.GetReaderInfo();

String Model = Info.GetModel();
String SerialNumber = Info.GetSerialNumber();
String FWRelease = MyReader.GetFWRelease();
```

GetReaderInfo and *GetFWRelease* methods are members of the *CAENRFIDReader* class.

Setting the power level

Most of the CAEN RFID readers allow to regulate the emitted power. This setting is useful in order to limit the read range of the reader, to limit the power as stated by the local regulations or for adapt the power depending on the antenna characteristics.

The API provides two methods, one for setting the power (*SetPower*) and one for getting the current power level (*GetPower*). The value passed to *SetPower* and returned by *GetPower* is expressed in milliWatt (mW) and refers to the power generated by the reader at the antenna's connectors.

The effective radiate power in mW ERP (*Perp*) is related to the conducted RF power (*Pw*) provided at the reader connector by the following formula:

$$Pw = \frac{Perp}{10^{\frac{(G-2.14-L)}{10}}}$$

where G is the antenna gain expressed in dBi and L the cable attenuation expressed in dB.

So, if you require (as often is) to set a ERP power level, the above formula has to be implemented in your software in order to obtain the conducted RF power.

Here below a code example that permits to obtain the desired ERP power from the antenna using the above formula and the *SetPower* method:

```
double Gain = 8.0;
double Loss = 1.5;
double ERPPower = 2000.0;
int OutPower;

OutPower = (int)(ERPPower/Math.Pow(10,((Gain-Loss-2.14)/10)));
MyReader.SetPower(OutPower);
```

and here a code example that permits to know the current setting of the ERP power using the inverse form of the formula and the *GetPower* method:

```
double Gain = 8.0;
double Loss = 1.5;
double ERPPower;
int OutPower;

OutPower = MyReader.GetPower();
ERPPower = ((double)power)*((double)Math.Pow(10,((Gain-Loss-2.14)/10)));
```

Typically CAEN RFID readers approximate automatically the power to the nearest available power level and the same is done for the minimum and maximum power level. For the available power levels please refer to the reader user manual.

There could be a difference between the power level set and the power level read just after the setting, this effect is normal and it is due to one or more of the following reasons:

- the set value was not exaclty one of the available power level on the reader so the real power level is the nearest available;
- the formula used to convert ERP to conducted power introduced a mathematical approximation;
- on some CAEN RFID readers the value returned by the *GetPower* method is a measurement of the power that can be affected by the accuracy of the measurement and the caracteristics of the connected antenna.

GetPower and *SetPower* methods are members of the *CAENRFIDReader* class.

Inventorying RFID tags

The fundamental operation of a UHF RFID system is the inventory of the population of tags inside the reading zone of the reader antennas. This operation, for the Gen2 protocol as for other UHF protocols, consists of a sequence of commands and replies exchanged between the reader and the tags, typically with stringent timings between them. CAEN RFID readers hide the complexity of the inventory algorithm implementing the algorithm in the firmware and providing a macro command as interface for the user.

The CAEN RFID API provides methods in order to activate the inventory process, the simplest method that just tries to collect all the tags inside the reading zone, a more complex one with a list of options and a method to start a cycle of inventories:

- Synchronous mode
- Event handling

Synchronous mode

Standard tag detection method (InventoryTag) is based on a polling mechanism: a call to the InventoryTag method/function results in a single read cycle and the detected tags in that cycle are returned.

InventoryTag method versions:

- InventoryTag Method (): simplest form; for more information see InventoryTag *Method ()* page 52
- InventoryTag Method (Byte[], Int16, Int16): with parameters *Mask, MaskLength, Position*; for more information see *InventoryTag Method (Byte[], Int16, Int16)* page 53
- InventoryTag Method (Byte[], Int16, Int16, Int16): with parameters *Mask, MaskLength, Position, Flag*; for more information see *InventoryTag Method (Byte[], Int16, Int16, Int16)* page 54
- InventoryTag Method (Int16, Byte[], Int16, Int16): with parameters *bank, Mask, MaskLength, Position*; for more information see *InventoryTag Method (Int16, Byte[], Int16, Int16)* page 55
- InventoryTag Method (Int16, Byte[], Int16, Int16, Int16): with parameters *bank, Mask, MaskLength, Position, Flag*; for more information see *InventoryTag Method (Int16, Byte[], Int16, Int16, Int16)* page 56

In the simplest form the inventory process can be activated simply by calling the *InventoryTag* method of the *CAENRFIDLogicalSource* class. This method has no parameters at all and returns an array of CAENRFIDTag objects once a complete run of the inventory algorithm is performed inside the reader (a CAENRFIDTag object is a software representation of the physical tag carrying the data associated to it like the EPC code, its length, the type of the tag and others). For a code sample look at the *Getting started with API* page 10 of this manual.

A more complete version of the *InventoryTag* methods takes parameters for filtering tags and to activate some optional features. The parameters used to filter the tags that have to be detected are: the memory bank (Bank), the mask (Mask), the length of the mask (MaskLength) and the start address (Position) for the matching. Using those parameters a Gen2 Select command is issued before starting the inventory process in order to match only the interesting tags. In the matching process the Mask parameter is compared to the memory bank content starting from the address Position for MaskLength bits. Only the matching tags will be involved in the inventory process and returned back to the user by the *InventoryTag* method.

The user can also choose the result of the matching mechanism, i.e. can choose to return the matching tags, the not matching tags or all the tags (ignoring indeed the filter). This setting can be changed using the *SetSelected_EPC_C1G2* method of the *CAENRFIDLogicalSource* class; possible values for its only parameter are:

- EPC_C1G2_SELECTED_YES: for matching tags;
- EPC_C1G2_SELECTED_NO: for non-matching tags;
- EPC_C1G2_SELECTED_ALL: for all tags (no filter).

An additional parameter (Flags) permits to activate special features of the inventory process, it is a bit mask where only the 5 less significant bits are used.

Bit 0 enables (1) or disables (0) the Return Signal Strength Indicator (RSSI) reading for each tag for those readers supporting it.

Bit 1 enables (1) or disables (0) the so called framed mode: if the framed mode is not enabled (default behavior) all the tags collected during the inventory process are stored into the reader memory and returned back to the user at the end of the process. With the framed mode enabled as soon as a tag is detected is returned immediately to the user. This behavior results in a better responsiveness of the application especially with large population of tags and it is suggested when a small embedded reader with

limited memory is used. It is mandatory to enable the framed bit when the continuous mode is enabled (see next bit description).

Bit 2 enables (1) or disables (0) the continuous mode: when this bit is enabled the reader implements internally a cycle of inventories. The number of executed inventories is determined by the ReadCycle parameter that can be set with the *SetReadCycle* method of the *CAENRFIDLogicalSource* class. When ReadCycle is 0 the cycle is repeated indefinitely until an abort command is sent to the reader.

Bit 3 enables (1) or disable(0) the compact data mode: when this flag is enabled, the inventory method will return back to the caller only the EPC code of the tag and all the other information like the timestamp and the type of the tag are filled with fake values. This flag is useful when it is necessary to reduce the data exchanged on the host interface, typically when the interface is slow (low baud rate serial interfaces).

Bit 4 enables (1) or disables (0) the readout of the TID during the inventory process.

A further option is to use an event-based inventory handling that means start a continuous and autonomous inventorying getting immediately the control of the thread flow to the caller. All the readings will be received by the application as software events. The user needs to define an event handler that will take care of handling the data coming from the tags. For more information on event-based inventory, please refer to the following paragraph.

Event Handling

While standard tag detection method (InventoryTag) is based on a polling mechanism (a call to the InventoryTag method/function results in a single read cycle and the detected tags in that cycle are returned), a useful variant ("continuous mode") uses an event mechanism to notify detected tags.

The methods involved in the “continuous mode” are:

- EventInventoryTag: with parameters *Mask, MaskLength, Position, Flag, InvParams -> pCallBack*; for more information see *EventInventoryTag Method* page 43
- InventoryAbort Method: for more information see *InventoryAbort Method* page 98
- ForceAbort Method: for more information see *ForceAbort Method* page 89

A call to the EventInventoryTag method/function starts a continuous tags detection algorithm (multiple read cycles) and an event is generated for each read cycle to notify the detected tags.

The user of the library can define an event handler method/function that is called automatically when the event raises; the data related to the event is passed to the handler as a parameter.

The user can define the number of read cycles that the EventInventoryTag have to perform using the ReadCycle parameter of the relevant LogicalSource. If ReadCycle is equal to 0 the EventInventoryTag method loops indefinitely.

The continuous mode is obtained by setting to 1 both framed (bit 1) and continuous (bit 2) flags.

The "continuous mode" can be interrupted using the InventoryAbort method function.

In readers equipped with button, if the event trigger flag (bit 5) is enabled and the continuous mode is enabled (bit 1 and bit 2), the event handler is recalled every time the button is pressed.

The event handling is implemented using the standard event handling mechanism in .NET and Java/Android while in C it is simulated using the callback mechanism.

No other methods can be invoked on logical source and reader, during the continuous mode, nor inside the event handler. The only operation allowed is an inventory abort, that must be used to stop a reader which is working in continuous mode.

InventoryTag and *EventInventoryTag* methods are members of the *CAENRFIDLogicalSource* class, while ForceAbort and *InventoryAbort* are members of the CAENRFIDReader Class.

EventInventoryTag Sample

Here below a code sample about the *EventInventoryTag* method: the code shows how to setup an event handler for processing data sent by the reader, start a continuous inventory operation and at the end stop it using the *InventoryAbort* method.

```
static void Main(string[] args)
{
 CAENRFIDReader Reader = new CAENRFIDReader();
 CAENRFIDLogicalSource LS0;
 byte[] Mask = new byte[4];

 Reader.CAENRFIDEvent += new CAENRFIDEventHandler(Reader_EventHandler);
 Reader.Connect(CAENRFIDPort.CAENRFID_TCP, "10.0.32.125");

 LS0 = Reader.GetSource("Source_0");

 LS0.SetReadCycle(0);
 LS0.EventInventoryTag(Mask, 0x0, 0x0, 0x06);
 Thread.Sleep(2*1000);
 Console.WriteLine("Main Task awake");
 Console.WriteLine("Tags read : " + ntag.ToString());
 Reader.InventoryAbort();
 Console.ReadLine();

 Reader.Disconnect();
}

static void Reader_EventHandler(object Sender, CAENRFIDEventArgs Event)
{
 foreach (CAENRFIDNotify n in Event.Data)
 {
  ntag++;
 }
}
```

In the readers equipped with button (for example in the R1240I qID reader) you can perform the inventory just by pressing the scan button (for more info, refer to the technical specifications of your reader), setting the flag parameter of the *EventInventoryTag* as follows:

```
LS0.EventInventoryTag (Mask, 0x0, 0x0, 0x26);
```

Note the use of the flag parameter in the method EventInventoryTag: enabling a continuous cycle is achieved through the activation of the FRAMED and CONTINUOUS fields while the suspension of the cycle by pressing the button requires the activation of the field TRIGGER EVENT.

Optimizing the inventory process

EPC Class1 Gen2 protocol defines a set of parameters useful for the optimization of the tags' detection, in the current paragraph we give a brief explanation of the most useful ones and we show how to work with those parameters using the CAEN RFID API.

The Q parameter

The Gen2 protocol inventory method is based on the so called “Slotted ALOHA” algorithm; without going into the details of the algorithm, it is important to know that it foresees a division of the time in discrete slots. Only one tag can be detected for each slot, if two or more tags reply on the same slot a collision is generated and the tags are not detected so a further iteration of the algorithm is needed.

The number of time slots is defined in the Gen2 protocol as 2Q where Q is a parameter ranging from 0 to 15 that can be set by the user.

The optimal Q value for a certain application is related to the average number of tags that are simultaneously present in the reading zone. A few tags require only a few slots, whereas many tags require many slots. Left to its own, the reader doesn’t have any way of knowing how many tags are under the antenna's field until it counts them, which may be difficult if its initial “guess” is wildly wrong. CAEN RFID reader will work faster and more efficiently if you provide an accurate starting value for Q corresponding to the expected tag population: the more are the collisions the worse are the performance of the reader in detecting the tags. The detection method implemented inside CAEN RFID readers has also an auto-adaptive mechanism in order to generate more or less time slots when needed but, starting with an adequate number of time slots, helps this mechaism to avoid wasting time.

For reference, the following table gives recommended values of Q that produce reasonably efficient inventories for varying numbers of tags in the read zone.

Estimated number of tags	Starting Q value
1	0
2	1
3 – 6	2
7 – 15	3
16 – 30	4
30 – 50	5
50 – 100	6
100 – 200	7

Tab. 3.1: Recommended Q values

CAEN RFID API provides the methods to set and read back the starting Q value: *SetQ_EPC_C1G2* and *GetQ_EPC_C1G2*.

Here below a code sample where the Q value is set to 3 before to start the inventory process:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using com.caen.RFIDLibrary;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            CAENRFIDReader MyReader = new CAENRFIDReader();

            MyReader.Connect(CAENRFIDPort.CAENRFID_RS232, "COM3");

            CAENRFIDLogicalSource MySource = MyReader.GetSource("Source_0");

            MySource.SetQ_EPC_C1G2(3);

            CAENRFIDTag[] MyTags = MySource.InventoryTag();

            if (MyTags.Length > 0)
            {
                for (int i = 0; i < MyTags.Length; i++)
                {
                    String s = BitConverter.ToString(MyTags[i].GetId());
                    Console.WriteLine(s);
                }
            }
            Console.WriteLine("Press a key to end the program.");
            Console.ReadKey();
            MyReader.Disconnect();
        }
    }
}
```

SetQ_EPC_C1G2 and *GetQ_EPC_C1G2* methods are members of the *CAENRFIDLogicalSource* class.

Sessions

When a tag is singulated by the “slotted ALOHA” algorithm described in the previous paragraph, it switches automatically from an internal state A to another internal state B.

In the EPC C1G2 terminology these states are called “target A” and “target B” respectively; for our purposes we can refer to target A as the non-inventoried state and to target B as the inventoried state: singulation makes therefore switch a tag from the non-inventoried state to the inventoried state.

By default, CAEN RFID readers look for non-inventoried tags so once a tag has been inventoried it is no more detectable by the reader until it returns back in the non-inventoried state.

Tags return in the non-inventoried state autonomously and the times needed to return detectable can be regulated using the session parameter. This tuning can have a big impact on the inventory process performances especially in case of large tags' population.

Gen2 protocol provides four different sessions: S0, S1, S2, S3. Each session has its own independent inventoried and non-inventoried states with its specific persistence time as shown in the following table:

Flag	Required persistence
S0 inventoried flag	Tag energized: Indefinite Tag not energized: None
S1 inventoried flag	Tag energized: • Nominal temperature range: 500ms < persistence < 5s • Extended temperature range: Not specified Tag not energized: • Nominal temperature range: 500ms < persistence < 5s • Extended temperature range: Not specified
S2 inventoried flag	Tag energized: Indefinite Tag not energized: • Nominal temperature range: 2s < persistence • Extended temperature range: Not specified
S3 inventoried flag	Tag energized: Indefinite Tag not energized: • Nominal temperature range: 2s < persistence • Extended temperature range: Not specified

Tab. 3.2: Persistence Time

When session S0 is used, each time the reader switches off the radiofrequency field, the tags return back in the original not-inventoried status so that they are again detectable with a successive inventory algorithm round. Since CAEN RFID readers switch off the field at the end of each inventory round, all the tags are detected again each time the inventory is repeated. This behavior can be desirable for testing purposes or when a certain level of redundancy is needed but it is typically not efficient and generate a lot of useless information.

With session S1 the tags, once singulated, remain in the inventoried state for a time in the range between 500ms and 5s regardless if the tag is energized or not; during this period they are no more detectable by the subsequent inventory iterations. On large tags' populations this behavior helps to reduce the number of tags simultaneously detectable by the reader optimizing the speed of the inventory process and reducing the generated data.

Sessions S2 and S3 have a longer and not explicitly limited persistence time giving the opportunity to detect tags only one time during the inventory process repetitions. These sessions are for advanced use only and are out of the scope of this manual.

CAEN RFID API provides the methods to set and read back the session parameter: *SetSession_EPC_C1G2* and *GetSession_EPC_C1G2*.

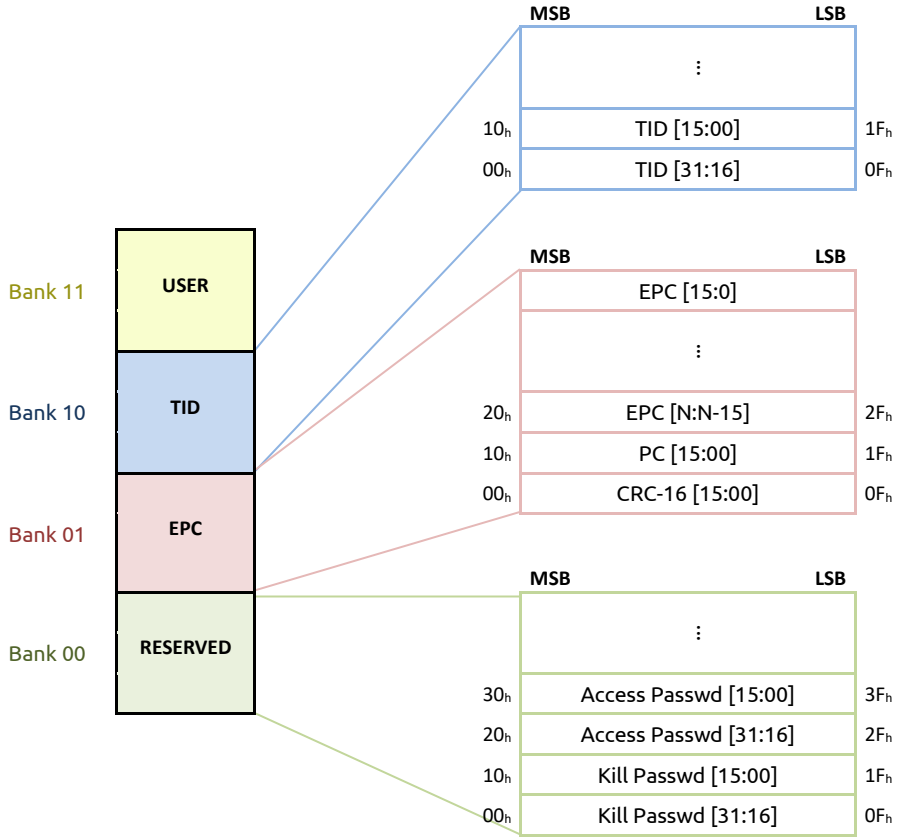
Here below a code sample where the session S1 is used during the inventory process:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using com.caen.RFIDLibrary;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            CAENRFIDReader MyReader = new CAENRFIDReader();

            MyReader.Connect(CAENRFIDPort.CAENRFID_RS232, "COM3");

            CAENRFIDLogicalSource MySource = MyReader.GetSource("Source_0");


MySource.SetSession_EPC_C1G2(CAENRFIDLogicalSourceConstants.EPC_C1G2_SESSION_S1);

            CAENRFIDTag[] MyTags = MySource.InventoryTag();

            if (MyTags.Length > 0)
            {
                for (int i = 0; i < MyTags.Length; i++)
                {
                    String s = BitConverter.ToString(MyTags[i].GetId());
                    Console.WriteLine(s);
                }
            }
            Console.WriteLine("Press a key to end the program.");
            Console.ReadKey();
            MyReader.Disconnect();
        }
    }
}
```

SetSession_EPC_C1G2 and *GetSession_EPC_C1G2* methods are members of the *CAENRFIDLogicalSource* class.

Reading and writing Gen2 tags

Gen2 tags contains a memory with the following structure:



The inventory process returns the EPC code that is part of the content of the EPC memory bank as a side-effect of the singulation process. All the memory content, anyway, can be read using the Gen2 Read command and the rewritable memory can be written using the Gen2 Write command.

The CAEN RFID API provides methods to read (*ReadTag_EPC_C1G2*) and write (*WriteTag_EPC_C1G2*) data into the tags' memory; the methods implement internally a Select command that is used to identify the tag on which execute the read or write command permitting to read/write into a specific tag even if multiple tags are inside the reading zone. The identification of the tag is executed by matching the EPC code that can be previously obtained by an inventory process. The other parameters needed to execute the read and write command are the memory bank, the starting address, the data length and, in case of the write command, the data that have to be written.

The API provides also a “secured” version of those commands that require a password as an additional parameter. These variants of the methods have to be used in the case the Access password is set into the tag. Tags with a non-zero Access password are “protected” (in Gen2 terminology: tags with a non-zero Access password are in Open state) and the user needs to know and passes to the tag the Access password in order to write into the tag memory (in Gen2 terminology: the tag has to switch in Secured state using the password before to allow to write).

Here below a code sample of reading/writing data from/to the user memory:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using com.caen.RFIDLibrary;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            CAENRFIDReader MyReader = new CAENRFIDReader();

            MyReader.Connect(CAENRFIDPort.CAENRFID_RS232, "COM3");

            CAENRFIDLogicalSource MySource = MyReader.GetSource("Source_0");

            CAENRFIDTag[] MyTags = MySource.InventoryTag();

            if (MyTags.Length > 0)
            {
                for (int i = 0; i < MyTags.Length; i++)
                {
                    String EPCString = BitConverter.ToString(MyTags[i].GetId());
                    Console.WriteLine(EPCString);

                    byte[] DataToWrite;
                    ASCIIEncoding Enc = new ASCIIEncoding();

                    DataToWrite = Enc.GetBytes("Hello!");
                    MySource.WriteTagData_EPC_C1G2(MyTags[i], 3, 0, 6, DataToWrite);
                    Console.WriteLine("Tag written!");

                    byte[] DataToRead;
                    DataToRead = MySource.ReadTagData_EPC_C1G2(MyTags[i], 3, 0, 6);
                    Console.WriteLine("Tag read, value = " + Enc.GetString(DataToRead));
                }


            }
            Console.WriteLine("Press a key to end the program.");
            Console.ReadKey();
            MyReader.Disconnect();
        }
    }
}
```

The read or write command will be executed on the first tag that replies to the reader chosen from those matching the filtering criterion.

ReadTag_EPC_C1G2 and Write*Tag_EPC_C1G2* methods are members of the *CAENRFIDLogicalSource* class.

Locking Gen2 tags

The EPCGlobal Class1 Gen2 protocol provides a mechanism to lock temporarily or permanently blocks of tag memory. The user can lock an entire memory bank with the only exception of the Reserved memory bank where it is allowed to lock independently the Access Password and the Kill Password. The lock mechanism prevents the possibility of further modifications on locked memory banks and, only for the passwords in the Reserved memory bank, it prevents also the possibility to read back the data (the passwords). The lock command works with a single parameter called *payload* that includes both the lock's type (permanent or not) and the bank to lock. The format of the payload is described by the following tables:

Lock-Command Payload

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Kill Mask		Access Mask		EPC Mask		TID Mask		User Mask		Kill Action		Access Action		EPC Action		TID Action		User Action	

Masks and Associated Action Fields

	Kill pwd		Access pwd		EPC memory		TID memory		User memory	
	0	1	2	3	4	5	6	7	8	9
Mask	skip/ write	skip/ write	skip/ write	skip/ write	skip/ write	skip/ write	skip/ write	skip/ write	skip/ write	skip/ write
	10	11	12	13	14	15	16	17	18	19
Action	pwd read/ write	perma lock	pwd read/ write	perma lock	pwd write	perma lock	pwd write	perma lock	pwd write	perma lock

Tab. 3.3: Lock payload and usage

pwd-write	permalock	Description
0	0	Associated memory bank is writeable from either the **open** or **secured** states
0	1	Associated memory bank is permanently writeable from either the **open** or **secured** states and may never be locked
1	0	Associated memory bank is writeable from the **secured** state but not from the **open** state
1	1	Associated memory bank is not writeable from any state
pwd-read/write	**permalock**	**Description**
0	0	Associated password location is readable and writeable from either the **open** or **secured** states
0	1	Associated password location is permanently readable and writeable from either the **open** or **secured** states and may never be locked
1	0	Associated password location is readable and writeable from **secured** state but not from **open** state
1	1	Associated password location is not readable or writeable from any state

Tab. 3.4: Lock Action-field functionality

The CAEN RFID API provides a method (*LockTag_EPC_C1G2*) to lock the tag memory contents that mimic exactly the behaviour of the protocol command. The payload parameter is implemented as a bitmask with the meaning described by the above tables. The API provides also the secured version of the *LockTag_EPC_C1G2* method to be used when the Access password is set.

Here below a code example of the *LockTag_EPC_C1G2* method utilization.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using com.caen.RFIDLibrary;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            CAENRFIDReader MyReader = new CAENRFIDReader();

            MyReader.Connect(CAENRFIDPort.CAENRFID_RS232, "COM3");

            CAENRFIDLogicalSource MySource = MyReader.GetSource("Source_0");

            CAENRFIDTag[] MyTags = MySource.InventoryTag();

            if (MyTags.Length > 0)
            {
                for (int i = 0; i < MyTags.Length; i++)
                {
                    String EPCString = BitConverter.ToString(MyTags[i].GetId());
                    Console.WriteLine(EPCString);

// +-------+-------+-------+-------+-------+-------+-------+-------+-------+-------+
// | Kill  | Acces | EPC   | TID   | User  | Kill  | Acces |  EPC  |  TID  |  User |
// | Mask  | Mask  | Mask  | Mask  | Mask  | Act.  | Act.  |  Act. |  Act. |  Act. |
// +---+---+---+---+---+---+---+---+---+---+---+---+-------+-------+-------+-------+
// | W | P | W | P | W | P | W | P | W | P | W | P | W | P | W | P | W | P | W | P |
// +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
// | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
// +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
//
// Non permanent locking of User Memory Bank
//
                    int Payload = 0x00802;
                    MySource.LockTag_EPC_C1G2(MyTags[i], Payload);

                }
            }
            Console.WriteLine("Press a key to end the program.");
            Console.ReadKey();
            MyReader.Disconnect();
        }
    }
```

LockTag_EPC_C1G2 method is a member of the *CAENRFIDLogicalSource* class.

Killing Gen2 tags

The EPCGlobal Class1 Gen2 protocol provides a way to kill tags, that means tags, after the kill process, are no more readable. This process is password protected: you can kill only tags with a non-zero kill password. So, once you have set the kill password using the *WriteTag_EPC_C1G2* method on the RESERVED memory bank at address 0, you can use the *KillTag_EPC_C1G2* method of the *CAENRFIDLogicalSource* class to kill the tag.

The tag that has to be killed is selected just by matching the complete EPC along with the Kill Password.

Differently from the previously described methods, there is not a secured version of the kill method since the kill command is always protected by a specialized kill password.

Handling General purpose Inputs/Ouputs (GPIOs)

Almost all CAEN RFID readers are provided along with some programmable GPIOs (for further details on GPIOs for a specific CAEN reader please refer to the correspondent reader manual).

On each GPIO you can perform two basic operations:

- Selects its direction (INPUT, OUTPUT).
- Sets its value (HIGH, LOW).

GPIO direction can be set and read by means of a 4 byte long bitmask used in conjunction with the *GetIODirection* and *SetIODirection* methods. Each bit in the bitmask represents the GPIO direction: a '0' value means INPUT, a '1' value means OUTPUT.

A 4 byte long bitmask in conjunction with GetIO and SetIO functions is also used to set/get GPIO's OUTPUT/INPUT values:

- Each upset bit in the bitmask sets the correspondent GPIO to a HIGH logic voltage.
- Each cleared bit in the bitmask sets the correspondent GPIO to a LOW level voltage.

High and low voltage value varies with the specific CAEN's reader used. Please refer to the specific CAEN reader manual to know the effective voltage values adopted.

Let's suppose we have a reader with 4 GPIOs available and we want to program GPIO0, GPIO1 as OUTPUT and GPIO2, GPIO3 as INPUT : the bitmask associated to this settings is 0000..0011b and the code is as follows:

```
int MyDirections = 0x3; //hex format

MyReader.SetIODirection(MyDirections);
```

Coming to the GetIO and SetIO methods, we can get GPIO's input status and set GPIOs' output status as follows:

```
int InputVal = 0x0;
int OutputVal = 0x2; //GPIO0 output value : 0,GPIO1 output value : 1

MyReader.GetIO(&InputVal);
MyReader.SetIO(OutputVal);
```

4 BUFFERED READING

Some CAEN RFID mobile readers come with an additional working profile called buffered mode. In this mode the reader is constantly waiting the pressure of the hardware button that triggers a single inventory, and then any tag found during the inventory is stored into the internal buffer.

This mode doesn't need the API interactions until the user application needs to view and download the data. In this case, the CAEN RFID API provides a subset of functions that can be called only in this mode: the buffered functions.

With this kind of functions the user application can do the following:

- Download all (or a subset of) the tags identified and stored in the internal buffer
- Query the count of the tag stored in the internal buffer.
- Clear all the tags stored into the internal buffer.

In Buffered mode many functions are disabled so the user application can't call them as usual, but some (like SetPower) are still enabled.

NOTE: Remember that the switching from a mode to another one requires the reader reboot, so it cannot be done via API. See your CAEN RFID reader tech manual to learn how to switch from the default easy2read mode to buffered mode and viceversa.

Warning: While the developer designs the user application, he/she should take in account the storage limits of the internal buffer of the reader, that strictly depends on the reader model itself.

GetBufferedData Method

GetBufferedData ()

Description:

This method returns all the Tags stored in reader buffer using all the ReadPoints belonging to the Source.

Return value:

An array of CAENRFIDTag objects detected.

Syntax:

C# representation:

```
public CAENRFIDTag[]        GetBufferedData()
```

Java and Android representation:

```
public CAENRFIDTag[]        GetBufferedData()
                            throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes          CAENRFID_GetBufferedData(
                            CAENRFIDHandle        handle,
                            char                  *source,
                            CAENRFIDTag           **Receive,
                            int                   *Size);
```

Swift representation:

```
func getBufferedData() throws -> [CAENRFIDTag]?
```

GetBufferedData (Int32, Int32)

Description:

This method returns all the Tags stored in reader buffer using all the ReadPoints belonging to the Source.

Return value:

An array of CAENRFIDTag objects detected.

Syntax:

C# representation:

```
public CAENRFIDTag[]        GetBufferedData (
                            int                 Address,
                            int                 Length)
```

Java and Android representation:

```
public CAENRFIDTag[]        GetBufferedData (
                            int                 Address,
                            int                 Length)
                            throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes          CAENRFID_GetBufferedDataRange(
                            CAENRFIDHandle      handle,
                            char                *source,
                            int                 address,
                            int                 length
                            CAENRFIDTag         **Receive,
                            int                 *Size);
```

Swift representation:

```
func getBufferedData(
                            address: UInt,
                            length: UInt
                            ) throws -> [CAENRFIDTag]?
```

GetBufferSize Method

Description:

This method gets the current number of records (tags) stored in the reader internal buffer.

Return value:

The current number of items stored in the internal's reader buffer.

Syntax:

C# representation:

```
public int          GetBufferSize()
```

Java and Android representation:

```
public int   GetBufferSize()
throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_GetBufferSize(
CAENRFIDHandle      handle,
unsigned int                *Size);
```

Swift representation:

```
func getBufferSize() throws -> UInt32
```

ClearBuffer Method

Description:

This method deletes all items stored in the internal buffer.

Return value:

The current number of items stored in the internal's reader buffer.

Syntax:

C# representation:

```
public int          ClearBuffer
```

Java and Android representation:

```
public int          ClearBuffer()
throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_ClearBuffer(
     CAENRFIDHandle        handle);
```

Swift representation:

```
func clearBuffer() throws
```

5 CAEN RFID API STRUCTURE

CAENRFID Classes

In .NET (henceforth C#), Java/Android and Swift languages, CAENRFID methods are divided into the following classes:

Classes	Description
CAENRFIDEventArgs	This class defines the CAENRFID event arguments.
CAENRFIDException	This class defines the CAEN RFID exceptions.
IDSTagData	This class represents data returned by tags based on IDS Chip SL900A.
CAENRFIDLogicalSource	The CAENRFIDLogicalSource class is used to create logical source objects. Logical source objects represent an aggregation of read points (antennas). Operations on the tags are performed using the logical source methods. In addition to the methods used to operate on the tags, the logical source class exports methods to configure the anticollision algorithm and to configure the composition of the logical source itself.
CAENRFIDNotify	This class defines the structure of a notification message.
CAENRFIDReader	The CAENRFIDReader class is used to create reader objects which permit to access to CAEN RFID readers' configuration and control commands.
CAENRFIDReaderInfo	The CAENRFIDReaderInfo class is used to create reader info objects. Reader info objects represent the information about the reader device (model and serial number).
CAENRFIDTag	This class is used to define objects representing the tags. These objects are used as return value for the inventory methods and as arguments for many tag access methods.
CAENRFIDNetworkInfo	This class represents the IP v4 settings of the ethernet connection of the CAEN RFID reader.

Tab. 5.1: CAENRFID classes

Each class contains the following methods:

Method	Description
CAENRFIDEventArgs Class	
getData	Returns the event object value.
CAENRFIDException Class	
getError	Gets the error string associated to the exception.
CAENRFIDIDSTagData Class	
getADError	Gets the error status of the A/D.
getRangeLimit	Gets the range limit parameter.
getSensorValue	Gets the value obtained by the sensor.
CAENRFIDLogicalSource Class	
AddReadPoint	Adds a read point to the logical source.
Authenticate_EPC_C1G2	This method allows an interrogator to perform tag, interrogator or mutual authentication. The generic nature of the authenticate command allows it to support a variety of cryptographic suites. The number of authenticate commands required to implement an authentication depends on the authentication type and on the chosen cryptographic suite.
BlockProgramID_EPC_C1G2	Overloaded. This method can be used to optimize a programming operation on an ISO18000-6C (EPC Class1 Gen2) tag ID space, writing data a block of bytes at time. It's an optional command, so before use it, please refer to the tag manual.
BlockWriteTagData_EPC_C1G2	Overloaded. This method can be used to optimize a writing

	operation on an ISO18000-6C (EPC Class1 Gen2) tag memory, writing data a block of bytes at time. It's an optional command, so before use it, please refer to the tag manual.
ClearBuffer	This method deletes all items stored in the internal buffer.
EM4325_GetSensorData	The GetSensorData command allows the reader to get the UID (if required)
EM4325_GetUID	sensor's tag information with a single command. Sensor's tag information can be related to the last acquired sample or to a new sample acquired by the tag after receiving this command but before replying to it.
EM4325_ResetAlarms	Overload. The GetUID command allows the reader to get the UID from a tag with a single command. The length and format of the UID is defined by the allocation class which shall be 0xE0(length: 64 bit), 0xE3(length : 80 bit), 0xE2 (length : 96 bit) or 0x44, 0x45, 0x46, 0x47 (length : 64).
EM4325_SendSPI	The ResetAlarms command allows the reader to reset/clear the alarm conditions
EventInventoryTag	A call to this method will start a sequence of read cycle on each read point linked to the logical source. The readings will be notified to the controller via event generation.
GetBufferedData	Overloaded. The function returns all the Tags stored in reader memory using all the ReadPoints belonging to the Source.
GetBufferSize	This method gets the current number of records (tags) stored in the reader internal buffer.
GetInventoryCounts	This method can be used to get the current setting for the number of inventory counts performed by the logical source after pressing the TRIGGER during the inventory algorithm execution.
GetInventoryDwellTime	This method can be used to get the inventory execution time (msec) used by the logical source during the inventory algorithm execution.
GetInventoryQuietTime	This method can be used to get the inventory quiet time (msec) used by the logical source during the inventory algorithm execution.
GetMaxQ_EPC_C1G2	This method can be used to retrieve the current maximum Q value (see EPC Class1 Gen2 protocol specification) used by the anticollision algorithm when called on this logical source.
GetMinQ_EPC_C1G2	This method can be used to retrieve the current minimum Q value (see EPC Class1 Gen2 protocol specification) used by the anticollision algorithm when called on this logical source.
GetName	Gets a string representing the name of the logical source.
GetNumMinQ_EPC_C1G2	This method can be used to get the number of inventory round count to perform with minimum EPC Class 1 Gen. 2 Q value and no tag has been found on each readpoint of this logical source, before stop the inventory execution.
GetQ_EPC_C1G2	This method can be used to retrieve the current setting for the initial Q value (see EPC Class1 Gen2 protocol specification) used by the anticollision algorithm when called on this logical source.
GetReadCycle	Gets the current setting for the number of read cycles performed by the logical source during the inventory algorithm execution.
GetSelected_EPC_C1G2	This method can be used to retrieve the Selected flag (see EPC Class1 Gen2 protocol specification) used by the anticollision algorithm when called on this logical source.
GetSession_EPC_C1G2	This method can be used to retrieve the Session setting (see EPC Class1 Gen2 protocol specification) used by the anticollision algorithm when called on this logical source.
GetTarget_EPC_C1G2	This method can be used to retrieve the Target setting (see EPC Class1 Gen2 protocol specification) used by the anticollision algorithm when called on this logical source.
InventoryTag	Overloaded. A call to this method will execute a read cycle on each read point linked to the logical source. Depending on the air protocol setting it will execute the appropriate anticollision algorithm.
isReadPointPresent	Checks if a read point is present in the logical source.

KillTag_EPC_C1G2	Overloaded. This method can be used to kill an EPC of an EPC Class 1 Gen 2 tag.
LockBlockPermaLock_EPC_C1G2	This method implements the BLockPermaLock with ReadLock=1 as specified in EPCC1G2 rev. 1.2.0 protocol.
LockTag_EPC_C1G2	Overloaded. This method can be used to lock a memory bank of an EPC Class 1 Gen 2 tag.
NXP_Calibrate	Overloaded. This method can be used to issue a Calibrate custom command as defined by the NXP G2XM and G2XL datasheet.
NXP_EnableBrandIdentifier	This method is used to enable/disable the select and return of Brand Identifier during subsequent inventory commands. Refer to NXP UCODE 8/8m datasheet
NXP_InventoryWithBrandIdentifier	This method is used to run an inventory with the NXP UCODE 8/8m Brand Identifier returning option.
ProgramID_EPC_C1G1	This method can be used to write the EPC of an EPC Class 1 Gen 1 tag.
ProgramID_EPC_C1G2	Overloaded. This method can be used to write the EPC of an EPC Class 1 Gen 2 tag.
ReadBLockPermalock_EPC_C1G2	This method implements the BLockPermaLock with ReadLock=0 as specified in EPCC1G2 rev. 1.2.0 protocol.
ReadTagData_EPC_C1G2	Overloaded. This method can be used to read a portion of memory in a ISO18000-6C (EPC Class1 Gen2) tag.
RemoveReadPoint	Removes a read point from the logical source.
SetInventoryCounts	This method can be used to set the current setting for the number of inventory counts performed by the logical source after pressing the TRIGGER during the inventory algorithm execution.
SetInventoryDwellTime	This method can be used to set the inventory execution time (msec) used by the logical source during the inventory algorithm execution.
SetInventoryQuietTime	This method can be used to set the inventory quiet time (msec) used by the logical source during the inventory algorithm execution.
SetMaxQ_EPC_C1G2	This method can be used to set the current maximum Q value (see EPC Class1 Gen2 protocol specification) used by the anticollision algorithm when called on this logical source.
SetMinQ_EPC_C1G2	This method can be used to set the current minimum Q value (see EPC Class1 Gen2 protocol specification) used by the anticollision algorithm when called on this logical source.
SetNumMinQ_EPC_C1G2	This method can be used to set the number of inventory round count to perform with minimum EPC Class 1 Gen. 2 Q value and no tag has been found on each readpoint of this logical source, before stop the inventory execution.
SetQ_EPC_C1G2	This method can be used to set the initial Q value (see EPC Class1 Gen2 protocol specification) used by the anticollision algorithm when called on this logical source.
SetReadCycle	Sets the number of read cycles to be performed by the logical source during the inventory algorithm execution.
SetSelected_EPC_C1G2	This method can be used to set the Session (see EPC Class1 Gen2 protocol specification) used by the anticollision algorithm when called on this logical source.
SetSession_EPC_C1G2	This method can be used to set the Session (see EPC Class1 Gen2 protocol specification) used by the anticollision algorithm when called on this logical source.
SetTarget_EPC_C1G2	This method can be used to set the Target setting (see EPC Class1 Gen2 protocol specification) used by the anticollision algorithm when called on this logical source.
SetTidLength	This method can be used to set the excepted TID length when TID reading is enabled as flag in inventory methods.
SL900A_GetBatteryLevel	The SL900A_GetBatteryLevel method try to read the level of the battery
SL900A_GetCalibrationData	The SL900A_GetCalibrationData method try to read the calibration data for the internal and external sensors. The 72-bits content of the Calibration data field and the SFE

	parameters field is returned as defined in the IDS SL900A datasheet.
SL900A_OpenArea	The SL900A_OpenArea method opens the specified area of the memory.
SL900A_SetPassword	The SL900A_SetPassword method sets the password for the specified memory area.
Untraceable_EPC_C1G2	This method allows an interrogator with an asserted untraceable privilege to instruct a Tag to (a) alter the L and U bits in EPC memory, (b) hide memory from interrogators with a deasserted Untraceable privilege and/or (c) reduce its operating range for all interrogators.
WriteTagData_EPC_C1G2	Overloaded. This method can be used to write a portion of memory in a ISO18000-6C (EPC Class1 Gen2) tag.
CAENRFIDNetworkInfo Class	
Address	Gets the CAEN RFID reader IPv4 address.
NetMask	Gets the CAEN RFID reader IPv4 netmask.
Gateway	Gets the CAEN RFID reader IPv4 gateway.
CAENRFIDNotify Class	
getDate	Returns a timestamp representing the time at which the event was generated.
getPC	Returns the tag PC code
getReadPoint	Returns the read point that has detected the tag.
getRSSI	Returns the RSSI value measured for the tag.
getStatus	Returns the event type associated to the tag.
getTagID	Returns the tag ID (the EPC code in Gen2 tags).
getTagLength	Returns the tag ID length.
getTagSource	Returns the name of the logical source that has detected the tag.
getTagType	Returns the air protocol of the tag.
getTID	Returns the TID field value in a EPC Class 1 Gen 2 Tag
getXPC	Returns the tag XPC words.
getFrequency	Returns the backscattering frequency expressed in KHz
getPhaseBegin	Returns the backscattering signal phase at the begin of the communication
getPhaseEnd	Returns the backscattering signal phase at the end of the communication
CAENRFIDReader Class	
Connect	Overloaded. Starts the communication with the reader. It must be called before any other call to method of the CAENRFIDReader object.
Disconnect	Closes the connection with the CAEN RFID Reader releasing all the allocated resources.
ForceAbort	This method tries to stop a pending continuos inventory (see EventInventoryTag method) that has not been stopped correclty by an InventoryAbort or Disconnect call. Choose the timeout value based on the expected reader load (large value if in presence of a large population of tags, small value if only few tags must be read).
GetBatteryLevel	This method gets the current battery charge.
GetBitRate	Gets the current setting of the RF bit rate.
GetFirmwareRelease	Overloaded. Permits to read the release of the firmware loaded into the device.
GetIO	Gets the current digital Input and Output lines status.
GetIODirection	Gets the current I/O direction setting as a bitmask. Each bit represents a I/O line, a value of 0 means that the line is configured as an input, 1 as an output. This setting has a meaning only for those readers with configurable I/O lines.
GetFHSSMode	This method gets the current Frequency Hopping status.
GetNetwork	Gets the IP v4 network settings of the reader (if supported).
GetPower	Gets the current setting of the RF power expressed in mW.

GetReaderInfo	Overloaded. Permits to read the reader information loaded into the device.
GetReadPointPower	Gets the conducted power in mW from antenna identified by the readpoint input parameter.
GetReadPoints	Gets the names of the read points (antennas) available in the reader.
GetReadPointStatus	Gets the CAENRFIDReadPointStatus object representing the status of a read point (antenna).
GetRFChannel	Gets the index of the RF channel currently in use. The index value meaning change for different country regulations.
GetRFRegulation	Gets the current RF regulation setting value.
GetSource	Gets a CAENRFIDLogicalSource object given its name
GetSourceNames	Gets the names of the logical sources available in the reader.
GetSources	Gets the CAENRFIDLogicalSource objects available on the reader.
InventoryAbort	Stops the EventInventoryTag execution.
MatchReadPointImpedance	Overloaded. This method matches the antenna impedance passed in ReadPoint.
SetBitRate	Sets the RF bit rate to use.
SetDateTime	Sets the Date/Time of the reader.
SetIO	Sets the Output lines value.
SetIODIRECTION	Sets the current I/O direction setting as a bitmask. Each bit represents a I/O line, a value of 0 means that the line is configured as an input, 1 as an output. This setting has a meaning only for those readers with configurable I/O lines.
SetReadPointPower	Set the conducted power in mW to the antenna identified by the readpoint input parameter.
SetPower	Sets the conducted RF power of the Reader.
SetRFChannel	Sets the RF channel to use. This method fixes the RF channel only when the listen before talk or the frequency hopping feature is disabled.
CAENRFIDReaderInfo Class	
GetModel	Gets the reader model.
GetSerialNumber	Gets the reader serial number.
CAENRFIDTag Class	
GetId	Returns the tag ID (the EPC code in Gen2 tags).
GetLength	Returns the tag ID length.
GetPC	Returns the tag PC code
GetReadPoint	Returns the read point that has detected the tag.
GetRSSI	Returns the RSSI value measured for the tag.
GetSource	Returns the name of the logical source that has detected the tag.
GetTID	Returns the tag TID (valid only for EPC Class 1 Gen 2 tags).
GetTimeStamp	Gets the Tag TimeStamp.
GetType	Returns the air protocol of the tag.
GetXPC	Returns the tag XPC words.
GetFrequency	Returns the backscattering frequency expressed in KHz
GetPhaseBegin	Returns the backscattering signal phase at the begin of the communication
GetPhaseEnd	Returns the backscattering signal phase at the end of the communication

Tab. 5.2: CAENRFID methods

CAENRFID Enumerations

The following enumerations are present in C# language. They correspond to classes in Java language and to enumerations and data types in C language:

Enumerations	Description
CAENRFIDBitRate	Gives a list of the supported radiofrequency profiles.
CAENRFIDLogicalSourceConstants	Gives a list of constants used for the configuration of the logical sources. Detailed explanation of the settings can be found in the EPC Class 1 Gen 2 and ISO 18000-6B specification documents.
CAENRFIDLogicalSource.InventoryFlag	Gives a list of constants used for the configuration of the inventory function.
CAENRFIDPort	Gives a list of the communication ports supported by the CAEN RFID readers.
CAENRFIDProtocol	Gives a list of the air protocol supported by the CAEN RFID readers.
CAENRFIDReadPointStatus	Gives a list of the possible ReadPoint status values.
CAENRFIDRFRegulations	The CAENRFIDRFRegulations gives a list of country radiofrequency regulations.
CAENRFIDRS232Constants	Gives a list of settings for the serial port configuration.
CAENRFIDTag.MemBanks	The CAENRFIDTag.MemBanks enumerates the bank name of a generic ISO18000-6C tag.

Tab. 5.3: CAENRFID Enumerations

6 METHODS DESCRIPTION

CAENRFIDEventArgs Class

The CAENRFIDEventArgs class defines the CAENRFID event arguments.

getData Method

Description:

This method returns the event object value.

Return value:

The value of the event object.

Syntax:

C# representation:

```
public CAENRFIDNotify[]            getData()
```

Java and Android representation:

```
public java.util.ArrayList         getData()
```

C representation:

Not supported

Swift representation:

Not supported

Data Accessory Method

This method returns the array of CAENRFIDTag in the event (default:1). It's nil if stream is end or there has been an issue during the continuous inventory.

Return value:

An array of CAENRFIDTag returned in the event.

Syntax:

C# representation:

Not supported

Java and Android representation:

Not supported

C representation:

Not supported

Swift representation:

```
Var data: [CAENRFIDTag]?
```

CAENRFIDException Class

The CAENRFIDException class defines the CAEN RFID exceptions.

getError Method

Description:

This method gets the error string associated to the exception.

Return value:

The string representing the error.

Syntax:

C# representation:

```
public string                    getError()
```

Java and Android representation:

```
public java.lang.String          getError()
```

C representation:

Not supported. This function does not exist in C language, see § *Error Handling* page 8 for more information.

Swift representation:

Not supported

CAENRFIDError Enumeration (Swift)

This enumeration defines the CAEN RFID error types thrown by Swift API methods, and it's an analog for the CAENRFIDException class.

communicationError(String errorMessage, CAENRFIDError? internalError)

Description:

This represents a communication error on main link. It's intended to be considered as a fatal error.

libraryError(String errorMessage)

Description:

This represents an error generated from the library, like pre-validation or initialization ones.

operationError(String errorMessage, UInt16 returnCode)

Description:

This error is the operation result code returned by the reader after the execution of a command, like a not supported command or parameter.

CAENRFID IDSTagData Class

This class represents data returned by tags based on IDS Chip SL900A.

In Java, Android, C# and Swift languages this class is composed by methods while in C language is represented by a struct (for more information see § *Overview on SDK* page 8):

C representation:

```
typedef struct {
                    BOOL                ADError_i;
                    unsigned int        RangeLimit_i;
                    unsigned int        SensorValue_i;
                    } CAENRFID_IDSTagData;
```

getADError Method

Description:

This method returns if an A/D error is raised.

Return value:

True if an A/D error occurs, false otherwise.

Syntax:

C# representation:

```
public bool ADError {
                    get;
                    }
```

Java and Android representation:

```
public boolean getADError()
```

C representation:

Not supported.

Swift representation:

```
public var ADError : Bool
```

getRangeLimit Method

Description:

This method returns the range limit set on sensor.

Return value:

A bitmask representing the range limit.

Syntax:

C# representation:

```
public uint RangeLimit      {
                            get;
                            }
```

Java and Android representation:

```
public int getRangeLimit()
```

C representation:

Not supported.

Swift representation:

```
public var rangeLimit : UInt
```

getSensorValue Method

Description:

This method returns the sensor value.

Return value:

A bitmask representing the value obtained by the sensor.

Syntax:

C# representation:

```
public uint SensorValue     {
                            get;
                            }
```

Java and Android representation:

```
public int getSensorValue()
```

C representation:

Not supported.

Swift representation:

```
public var sensorValue : UInt
```

CAENRFIDLogicalSource Class

The CAENRFIDLogicalSource class is used to create logical source objects. Logical source objects represent an aggregation of read points (antennas). Operations on the tags are performed using methods belonging to the logical source. In addition to the methods used to operate on the tags, the logical source class exports methods to configure the anticollision algorithm and to configure the composition of the logical source itself.

AddReadPoint Method

Description:

This method adds a read point to the logical source.

Parameters:

Name	Description
ReadPoint	A string representing the name of the read point (antenna).

Syntax:

C# representation:

```
public void  AddReadPoint(
                            string                  ReadPoint)
```

Java and Android representation:

```
public void  AddReadPoint(
                            java.lang.String        ReadPoint)
                            throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_AddReadPoint(
                            CAENRFIDHandle          handle,
                            char                    *SourceName,
                            char                    *ReadPoint);
```

Swift representation:

```
func addReadPoint(_ readPoint: String) throws
```

Authenticate_EPC_C1G2 Method

Warning: This method is not supported by all readers. For more information, please contact the support at *support@caenrfid.com*.

Description:

This method allows an interrogator to perform tag, interrogator or mutual authentication. The generic nature of the authenticate command allows it to support a variety of cryptographic suites. The number of authenticate commands required to implement an authentication depends on the authentication type and on the chosen cryptographic suite.

Parameters:

Name	Description
Tag	The CAENRFIDTag representing the tag to be authenticated.
senRep	Specifies whether a tag backscatters its response or first stores the response in its ResponseBuffer and then returns the data from there.
incRepLen	Specifies whether a tag omits or includes the response length in its reply
csi	Selects the cryptographic suite that tag and interrogator use for the authentication.
challenge	It includes parameters and data for authentication.
repLen	Specify the byte's length of the tag response.
password	The access password

Return value:

A byte array containing the tag response to the authenticate command.

Syntax:

C# representation:

```
public void                 Authenticate_EPC_C1G2(
                                CAENRFIDTag         Tag,
                                bool                senRep,
                                bool                incRepLen,
                                char                csi,
                                byte[]              challenge,
                                short               repLen,
                                uint                password)
```

Java and Android representation:

```
public void                 Authenticate_EPC_C1G2(
                                CAENRFIDTag         Tag,
                                boolean             senRep,
                                boolean             incRepLen,
                                char                csi,
                                byte[]              challenge,
                                short               repLen,
                                int                 password)
                            throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes          CAENRFID_Authenticate_EPC_C1G2(
                                CAENRFIDHandle      handle,
                                char*               SourceName,
                                CAENRFIDTag         *Tag,
                                BOOL                senRep,
                                BOOL                incRepLen,
                                char                csi,
                                byte                *challenge,
                                short               challengeLen,
                                short               repLen,
                                int                 password,
                                byte                **authResponse);
```

Swift representation:

```
func authenticate_EPC_C1G2(
                            tag: CAENRFIDTag,
                            senRep: Bool,
                            incRepLen: Bool,
                            csi: UInt8,
                            challenge: [UInt8],
                            repLen: UInt16,
                            accessPassword: UInt32
                            ) throws -> [UInt8]?
```

BlockProgramID_EPC_C1G2 Method

Warning: This method is not supported by all readers. For more information, please contact the support at *support@caenrfid.com*.

BlockProgramID_EPC_C1G2 Method (CAENRFIDTag, Int16, Int16)

Description:

This method can be used to optimize a programming operation on an ISO18000-6C (EPC Class1 Gen2) tag ID space, writing data a block of bytes at time. It's an optional command, so before use it, please refer to the tag manual.

Parameters:

Name	Description
Tag	The CAENRFIDTag representing the tag to be programmed, the ID contained in this object will be programmed into the tag
NSI	The Numbering System Identifier as defined in EPC Class 1 Gen 2 protocol specifications.
BlockLength	The length in byte of the single block of data.

Syntax:

C# representation:

```
public void  BlockProgramID_EPC_C1G2(
                    CAENRFIDTag           Tag,
                    short                 NSI,
                    short                 BlockLength)
```

Java and Android representation:

```
public void  BlockProgramID_EPC_C1G2(
                    CAENRFIDTag           Tag,
                    short                 NSI,
                    short                 BlockLength)
                    throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID  BlockProgramID EPC C1G2(
                    CAENRFIDHandle        handle,
                    CAENRFIDTag           *Tag,
                    short                 NSI,
                    short                 BlockLength);
```

Swift representation:

```
func blockProgramID_EPC_C1G2(
                    tag: CAENRFIDTag,
                    nsi: UInt16 = 0,
                    blockLength: UInt16
                    ) throws
```

BlockProgramID_EPC_C1G2 Method (CAENRFIDTag, Int16, Int16, Int32)

Description:

This method can be used to optimize a programming operation on an ISO18000-6C (EPC Class1 Gen2) tag ID space, writing data a block of bytes at time. It's an optional command, so before use it, please refer to the tag manual. The Custom command is executed after an Access command to switch the tag in the Secured state using the provided password.

Parameters:

Name	Description
Tag	The CAENRFIDTag representing the tag to be programmed, the ID contained in this object will be programmed into the tag
NSI	The Numbering System Identifier as defined in EPC Class 1 Gen 2 protocol specifications.
BlockLength	The length in byte of the single block of data.
AccessPassword	The access password

Syntax:

C# representation:

```
public void  BlockProgramID_EPC_C1G2(
                    CAENRFIDTag           Tag,
                    short                 NSI,
                    short                 BlockLength,
                    int                   AccessPassword)
```

Java and Android representation:

```
public void  BlockProgramID_EPC_C1G2(
                    CAENRFIDTag           Tag,
                    short                 NSI,
                    short                 BlockLength,
                    int                   AccessPassword)
                    throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_ BlockProgramID_EPC_C1G2(
                    CAENRFIDHandle        handle,
                    CAENRFIDTag           *Tag,
                    short                 NSI,
                    short                 BlockLength
                    int                   AccessPassword);
```

Swift representation:

```
func         blockProgramID_EPC_C1G2(
                    Tag: CAENRFIDTag,
                    NSI: UInt16 = 0,
                    BlockLength: UInt16,
                    AccessPassword: UInt32
                    ) throws
```

BlockWriteTagData_EPC_C1G2 Method

Warning: This method is not supported by all readers. For more information, please contact the support at *support@caenrfid.com*.

BlockWriteTagData_EPC_C1G2 Method (CAENRFIDTag, Int16, Int16, Int16, Int16, Byte[])

Description:

This method can be used to optimize a writing operation on an ISO18000-6C (EPC Class1 Gen2) tag memory, writing data a block of bytes at time. It's an optional command, so before use it, please refer to the tag manual.

Parameters:

Name	Description
Tag	The CAENRFIDTag representing the tag to be written.
MemBank	The memory bank where to write the data.
Address	The address where to start writing the data.
BlockLength	The length in byte of the single block of data.
Length	The number of byte to be written.
Data	An array of bytes representing the data to be written into the tag.

Syntax:

C# representation:

```
public void  BlockWriteTagData_EPC_C1G2(
                    CAENRFIDTag Tag,
                    short MemBank,
                    short Address,
                    short blockLength,
                    short Length,
                    byte[] Data)
```

Java and Android representation:

```
public void  BlockWriteTagData_EPC_C1G2(
                    CAENRFIDTag Tag,
                    short MemBank,
                    short Address,
                    short blockLength,
                    short Length,
                    byte[] Data)
                    throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_ BlockWriteTagData_EPC_C1G2(
                    CAENRFIDHandle        handle,
                    CAENRFIDTag           *Tag,
                    short                 MemBank,
                    Short                 Address,
                    short                 BlockLength,
                    short                 Length);
```

Swift representation:

```
func         blockWriteTagData_EPC_C1G2(
                    tag: CAENRFIDTag,
                    bank: CAENRFIDTag.MemBanks,
                    address: UInt16,
                    blockLength: UInt16,
                    length: UInt16,
                    data: [UInt8]) throws
```

BlockWriteTagData_EPC_C1G2 Method (CAENRFIDTag, Int16, Int16, Int16, Int16, Byte[], Int32)

Description:

This method can be used to optimize a writing operation on an ISO18000-6C (EPC Class1 Gen2) tag memory, writing data a block of bytes at time. It's an optional command, so before use it, please refer to the tag manual.

Parameters:

Name	Description
Tag	The CAENRFIDTag representing the tag to be written.
MemBank	The memory bank where to write the data.
Address	The address where to start writing the data.
BlockLength	The length in byte of the single block of data.
Length	The number of byte to be written.
Data	An array of bytes representing the data to be written into the tag.
AccessPassword	The access password

Syntax:

C# representation:

```
public void BlockWriteTagData_EPC_C1G2(
      CAENRFIDTag Tag,
      short MemBank,
      short Address,
      short blockLength,
      short Length,
      byte[] Data,
      int accessPassword
)
```

Java and Android representation:

```
public void BlockWriteTagData_EPC_C1G2(
      CAENRFIDTag Tag,
      short MemBank,
      short Address,
      short blockLength,
      short Length,
      byte[] Data,
      int accessPassword
)
                    throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_ BlockWriteTagData_EPC_C1G2(
                    CAENRFIDHandle        handle,
                    CAENRFIDTag           *Tag,
                    short                 MemBank,
                    Short                 Address,
                    short                 BlockLength,
                    short                 Length,
                    int                   AccessPassword);
```

Swift representation:

```
func         blockWriteTagData_EPC_C1G2(
                    tag: CAENRFIDTag,
                    bank: CAENRFIDTag.MemBanks,
                    address: UInt16,
                    blockLength: UInt16,
                    length: UInt16,
                    data: [UInt8],
                    accessPassword: UInt32
                    ) throws
```

ClearBuffer Method

> **Warning**: This method is supported only by mobile readers

Description:

This method deletes all items stored in the internal buffer.

Return value:

The current number of items stored in the internal's reader buffer.

Syntax:

C# representation:

```
public int          ClearBuffer
```

Java and Android representation:

```
public int          ClearBuffer()
throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_ClearBuffer(
     CAENRFIDHandle        handle);
```

Swift representation:

```
func clearBuffer() throws
```

EventInventoryTag Method

EventInventoryTag(byte[],Int16,Int16,Int16)

Description:

A call to this method will start a sequence of read cycle on each read point linked to the logical source. The readings will be notified to the controller via event generation.

Parameters:

Name	Description
Mask	A byte array representing the bitmask to apply.
MaskLength	A value representing the bit-oriented length of the bitmask.
Position	A value representing the first bit where the match will start.
Flag	A bitmask representing the InventoryTag options.
InvParams -> pCallBack	The user defined handler called by EventInventoryTag (only in C language).

Return value:

A boolean value that represents the status of the command: true if the reader has accepted the command; false otherwise.

Syntax:

C# representation:

```
public bool  EventInventoryTag(
                          byte[]                    Mask,
                          short                     MaskLength,
                          short                     Position,
                          short                     Flag)
```

Java and Android representation:

```
public boolean      EventInventoryTag(
                          byte[]                    Mask,
                          short                     MaskLength,
                          short                     Position,
                          short                     Flag)
                          throws CAENRFIDException
```

C representation:

```
typedef struct {
                    char                                *SourceName;
                    char                                *Mask;
                    unsigned char                       MaskLength;
                    unsigned char                       Position;
                    CAENRFID_INVENTORY_CALLBACK         pCallBack;
                    short                               flag;}
                    CAENRFID_EventInventoryParams;

CAENRFIDErrorCodes  CAENRFID_EventInventoryTag (
                    CAENRFIDHandle                      handle,
                    CAENRFID_EventInventoryParams       InvParams);
```

Swift representation:

```
func eventInventoryTag(
                    mask: [UInt8] = [],
                    maskLength: UInt16 = 0,
                    position: UInt16 = 0,
                    flag: UInt16 = CAENRFIDLogicalSource.InventoryFlag.FRAMED.rawValue + C
                    AENRFIDLogicalSource.InventoryFlag.CONTINUOUS.rawValue
                    ) throws -> Bool
```

GetBufferedData Method

Warning: This method is supported only by mobile readers

GetBufferedData ()

Description:

This method returns all the Tags stored in reader buffer using all the ReadPoints belonging to the Source.

Return value:

An array of CAENRFIDTag objects detected.

Syntax:

C# representation:

```
public CAENRFIDTag[]        GetBufferedData()
```

Java and Android representation:

```
public CAENRFIDTag[]        GetBufferedData()
                            throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes          CAENRFID_GetBufferedData(
                            CAENRFIDHandle          handle,
                            char                    *source,
                            CAENRFIDTag             **Receive,
                            int                     *Size);
```

Swift representation:

```
func getBufferedData() throws -> [CAENRFIDTag]?
```

GetBufferedData (Int32, Int32)

Description:

This method returns all the Tags stored in reader buffer using all the ReadPoints belonging to the Source.

Return value:

An array of CAENRFIDTag objects detected.

Syntax:

C# representation:

```
public CAENRFIDTag[]        GetBufferedData (
                            int                 Address,
                            int                 Length)
```

Java and Android representation:

```
public CAENRFIDTag[]        GetBufferedData (
                            int                 Address,
                            int                 Length)
                            throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes          CAENRFID_GetBufferedDataRange(
                            CAENRFIDHandle      handle,
                            char                *source,
                            int                 address,
                            int                 length
                            CAENRFIDTag         **Receive,
                            int                 *Size);
```

Swift representation:

```
func getBufferedData(
                            address: UInt,
                            length: UInt
                            ) throws -> [CAENRFIDTag]?
```

GetBufferSize Method

Warning: This method is supported only by mobile readers

Description:

This method gets the current number of records (tags) stored in the reader internal buffer.

Return value:

The current number of items stored in the internal's reader buffer.

Syntax:

C# representation:

```
public int          GetBufferSize()
```

Java and Android representation:

```
public int   GetBufferSize()
throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_GetBufferSize(
CAENRFIDHandle      handle,
unsigned int                *Size);
```

Swift representation:

```
func getBufferSize() throws -> UInt32
```

GetInventoryCounts Method

Description:

This method gets the current setting for the number of inventory counts performed by the logical source after pressing the TRIGGER during the inventory algorithm execution.

Return value:

The number of read cycles.

Syntax:

C# representation:

```
public int                GetInventoryCounts()
```

Java and Android representation:

```
public int                GetInventoryCounts()
                  throws  CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes        CAENRFID_GetInventoryCounts(
                                CAENRFIDHandle      handle,
                                char                *SourceName,
                                int                 *value);
```

Swift representation:

Not supported

GetInventoryDwellTime Method

Description:

This method gets the inventory execution time (msec) used by the logical source during the inventory algorithm execution.

Return value:

The inventory exceution time (msec).

Syntax:

C# representation:

```
public int                GetInventoryDwellTime();
```

Java and Android representation:

```
public int                GetInventoryDwellTime()
                  throws  CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes        CAENRFID_GetInventoryDwellTime(
                          CAENRFIDHandle            handle,
                          char                      *SourceName,
                          int                       *Value);
```

Swift representation:

Not supported

GetInventoryQuietTime Method

Description:

This method can be used to get the inventory quiet time (msec) used by the logical source during the inventory algorithm execution.

Return value:

The quite time expressed in ms.

Syntax:

C# representation:

```
public int                  GetInventoryQuietTime();
```

Java and Android representation:

```
public int                  GetInventoryQuietTime()
                    throws  CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes          CAENRFID_GetInventoryQuietTime(
                                    CAENRFIDHandle      handle,
                                    char                *SourceName,
                                    int                 *value);
```

Swift representation:

Not supported

GetMaxQ_EPC_C1G2 Method

Description:

This method can be used to retrieve the current maximum Q value (see EPC Class1 Gen2 protocol specification) used by the anticollision algorithm when called on this logical source.

Return value:

The current maximum Q value.

Syntax:

C# representation:

```
public int                      GetMaxQ_EPC_C1G2();
```

Java and Android representation:

```
public int                  GetMaxQ_EPC_C1G2()
                    throws  CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes          CAENRFID_GetMaxQ_EPC_C1G2(
                                    CAENRFIDHandle      handle,
                                    char                *SourceName,
                                    int                 *value);
```

Swift representation:

```
func GetMaxQ_EPC_C1G2() throws -> UInt
```

GetMinQ_EPC_C1G2 Method

Description:

This method can be used to retrieve the current minimum Q value (see EPC Class1 Gen2 protocol specification) used by the anticollision algorithm when called on this logical source.

Return value:

The current minimum Q value.

Syntax:

C# representation:

```
public int                      GetMinQ_EPC_C1G2();
```

Java and Android representation:

```
public int              GetMinQ_EPC_C1G2()
                  throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes      CAENRFID_GetMinQ_EPC_C1G2(
                              CAENRFIDHandle      handle,
                              char                *SourceName,
                              int                 *value);
```

Swift representation:

```
func GetMinQ_EPC_C1G2() throws -> UInt
```

GetName Method

Description:

This method gets a string representing the name of the logical source.

Return value:

A string representing the name of the logical source.

Syntax:

C# representation:

```
public string               GetName()
```

Java and Android representation:

```
public java.lang.String     GetName()
```

C representation:

Not supported.This function does not exist in C language, see § *Overview on SDK* page 7 for more information.

Swift representation:

Not supported

GetNumMinQ_EPC_C1G2 Method

Description:

This method can be used to get the inventory round count to perform with minimum EPC Class 1 Gen. 2 Q value when no tag has been found on each readpoint of this logical source, before stop the inventory execution.

Return value:

The current inventory round count with minimum Q.

Syntax:

C# representation:

```
public int                  GetNumMinQ_EPC_C1G2();
```

Java and Android representation:

```
public int          GetNumMinQ_EPC_C1G2()
throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_GetNumMinQ_EPC_C1G2(
                    CAENRFIDHandle        handle,
                    char                  *SourceName,
                    int                   *value);
```

Swift representation:

```
func GetMaxQ_EPC_C1G2() throws -> UInt
```

GetQ_EPC_C1G2 Method

Description:

This method can be used to retrieve the current setting for the initial Q value (see EPC Class1 Gen2 protocol specification) used by the anticollision algorithm when called on this logical source.

Return value:

The current initial Q value setting.

Syntax:

C# representation:

```
public int          GetQ_EPC_C1G2()
```

Java and Android representation:

```
public int          GetQ_EPC_C1G2()
            throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_GetQValue_EPC_C1G2(
                           CAENRFIDHandle        handle,
                           char                  *SourceName,
                           int                   *Q);
```

Swift representation:

```
func getQ_EPC_C1G2() throws -> UInt
```

GetReadCycle Method

Description:

This method gets the current setting for the number of read cycles performed by the logical source during the inventory algorithm execution.

ReadCycle affects only inventory performed with continuous mode (see § *EventInventoryTag Method* page 43).

Return value:

The number of read cycles.

Syntax:

C# representation:

```
public int          GetReadCycle()
```

Java and Android representation:

```
public int          GetReadCycle()
                    throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_GetReadCycle(
                    CAENRFIDHandle        handle,
                    char                  *SourceName,
                    int                   *value);
```

Swift representation:

```
func getReadCycle() throws -> UInt
```

GetSelected_EPC_C1G2 Method

Description:

This method can be used to retrieve the Selected flag (see EPC Class1 Gen2 protocol specification) used by the anticollision algorithm when called on this logical source.

Return value:

The current Selected value

Syntax:

C# representation:

```
public CAENRFIDLogicalSourceConstants     GetSelected_EPC_C1G2()
```

Java and Android representation:

```
public CAENRFIDLogicalSourceConstants     GetSelected_EPC_C1G2()
                                          throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes                        CAENRFID_GetSelected_EPC_C1G2(
                                          CAENRFIDHandle               handle,
                                          char                         *SourceName,
                                          CAENRFIDLogicalSourceConstants      *value);
```

Swift representation:

```
func getSelected_EPC_C1G2() throws -> CAENRFIDLogicalSourceConstants
```

GetSession_EPC_C1G2 Method

Description:

This method can be used to retrieve the Session setting (see EPC Class1 Gen2 protocol specification) used by the anticollision algorithm when called on this logical source.

Return value:

The current Session value setting.

Syntax:

C# representation:

```
public CAENRFIDLogicalSourceConstants     GetSession_EPC_C1G2()
```

Java and Android representation:

```
public CAENRFIDLogicalSourceConstants     GetSession_EPC_C1G2()
                                          throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_GetSession_EPC_C1G2(
                    CAENRFIDHandle                      handle,
                    char                                *SourceName,
                    CAENRFIDLogicalSourceConstants      *value);
```

Swift representation:

```
func getSession_EPC_C1G2() throws -> CAENRFIDLogicalSourceConstants
```

GetTarget_EPC_C1G2 Method

Description:

This method can be used to retrieve the Target setting (see EPC Class1 Gen2 protocol specification) used by the anticollision algorithm when called on this logical source.

Return value:

The current Target value setting.

Syntax:

C# representation:

```
public CAENRFIDLogicalSourceConstants     GetTarget_EPC_C1G2()
```

Java and Android representation:

```
public CAENRFIDLogicalSourceConstants     GetTarget_EPC_C1G2()
                                          throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes                CAENRFID_GetTarget_EPC_C1G2(
                                  CAENRFIDHandle                      handle,
                                  char                                *SourceName,
                                  CAENRFIDLogicalSourceConstants      *value);
```

Swift representation:

```
func getTarget_EPC_C1G2() throws -> CAENRFIDLogicalSourceConstants
```

GetTIDLength Method

Description:

This method can be used to get the excepted TID length when TID reading is enabled as flag in inventory methods.

Return value:

The The maximum length of the TID bank to read expressed in bytes.

Syntax:

C# representation:

```
public int          GetTIDLength()
```

Java and Android representation:

```
public int          GetTIDLength ()
                    throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_GetTIDLength(
                    CAENRFIDHandle        handle,
                    char                  *SourceName,
                    int                   *value);
```

Swift representation:

```
func GetTIDLength() throws -> UInt
```

InventoryTag Method

InventoryTag Method ()

Description:

A call to this method will execute a read cycle on each read point linked to the logical source. Depending on the air protocol setting it will execute the appropriate anticollision algorithm.

Return value:

An array containing the CAENRFIDTag objects representing the tags read from the read points.

Syntax:

C# representation:

```
public CAENRFIDTag[]        InventoryTag()
```

Java and Android representation:

```
public CAENRFIDTag[]        InventoryTag()
                            throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes          CAENRFID_InventoryTag (
                            CAENRFIDHandle        handle,
                            char                  *SourceName,
                            CAENRFIDTag           **Receive,
                            int                   *Size);
```

Swift representation:

```
func inventoryTag() throws -> [CAENRFIDTag]?
```

InventoryTag Method (Byte[], Int16, Int16)

Description:

A call to this method will execute a read cycle on each read point linked to the logical source.

Parameters:

Name	Description
Mask	A byte array representing the bitmask to apply.
MaskLength	A value representing the bit-oriented length of the bitmask.
Position	A value representing the first bit of ID where the match will start.

Return value:

An array containing the CAENRFIDTag objects representing the tags read from the read points.

Syntax:

C# representation:

```
public CAENRFIDTag[]        InventoryTag(
                            byte[]                Mask,
                            short                 MaskLength,
                            short                 Position)
```

Java and Android representation:

```
public CAENRFIDTag[]        InventoryTag(
                            byte[]                Mask,
                            short                 MaskLength,
                            short                 Position)
                            throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_FilteredInventoryTag(
                            CAENRFIDHandle        handle,
                            char                  *SourceName,
                            char                  *Mask,
                            unsigned char         MaskLength,
                            unsigned char         Position,
                            CAENRFIDTag           **Receive,
                            int                   *Size);
```

Swift representation:

```
public CAENRFIDTag[]        InventoryTag(
                            byte[]                Mask,
                            short                 MaskLength,
                            short                 Position)
                            throws CAENRFIDException
```

Remarks:

Depending on the air protocol setting it will execute the appropriate anticollision algorithm. This version of the method permits to specify a bitmask for filtering tag populations as described by the EPC Class1 Gen2 (ISO18000-6C) air protocol. The filtering will be performed on the memory bank specified by bank parameter, starting at the bit indicated by the Position index and for a MaskLength length. The method will return only the tags that match the given Mask. Passing a zero value for MaskLength it performs as the non-filtering InventoryTag method.

InventoryTag Method (Byte[], Int16, Int16, Int16)

Description:

A call to this method will execute a read cycle on each read point linked to the logical source.

Parameters:

Name	Description
Mask	A byte array representing the bitmask to apply.
MaskLength	A value representing the bit-oriented length of the bitmask.
Position	A value representing the first bit of ID where the match will start.
Flag	A bitmask representing the InventoryTag options.

Return value:

An array containing the CAENRFIDTag objects representing the tags read from the read points.

Syntax:

C# representation:

```
public CAENRFIDTag[]        InventoryTag(
                            byte[]                Mask,
                            short                 MaskLength,
                            short                 Position,
                            short                 Flag)
```

Java and Android representation:

```
public CAENRFIDTag[]        InventoryTag(
                            byte[]                Mask,
                            short                 MaskLength,
                            short                 Position,
                            short                 Flag)
                            throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_FlagInventoryTag (
                            CAENRFIDHandle        handle,
                            char                  *SourceName,
                            char                  *Mask,
                            unsigned char         MaskLength,
                            unsigned char         Position,
                            unsigned char         Flag,
                            CAENRFIDTag           **Receive,
                            int                   *Size);
```

Swift representation:

```
func inventoryTag(
    mask: [UInt8] = [],
    maskLength: UInt16 = 0,
    position: UInt16 = 0,
    flag: UInt16 = 0
) throws -> [CAENRFIDTag]?
```

Remarks:

Depending on the air protocol setting it will execute the appropriate anticollision algorithm. This version of the method permits to specify a bitmask for filtering tag populations as described by the EPC Class1 Gen2 (ISO18000-6C) air protocol. The filtering will be performed on the memory bank specified by bank parameter, starting at the bit indicated by the Position index and for a MaskLength length. The method will return only the tags that match the given Mask. Passing a zero value for MaskLength it performs as the non-filtering InventoryTag method. The Flags parameter permits to set InventoryTag method's options.In this case bit 1 and 2 of the flag (continuous and framed mode) are ignored.

Flag value meaning	
Bit 0	RSSI: a 1 value indicates that the reader will transmit the RSSI (Return Signal Strength Indicator) in the response.
Bit 3	Compact data: a 1 value indicates that only the EPC of the tag will be returned by the reader, a 0 value indicates that the complete data will be returned. In case that the compact option is enabled all the other data will be populated by this library with fakes values.
Bit 4	TID reading: a 1 value indicates that also the TID of the tag will be returned by the reader together with the other information.
Bit 5	Event trigger: when this flag is set together with the continuous acquisition flag, the inventory cycle is performed in the same way of the continuous mode with the only difference that the inventory command is sent only by pressing the button (only for readers equipped with button).
Bit 6	XPC: a 1 value allows the reader to get the XPC word if backscattered by a tag.Tags that do not backscatter the XPC words will return an XPC array with all the 4 bytes set to 0
Bit 8	PC: a 1 value allows the reader to return the PC of a Gen2 tag in addition to the ID.
Bit 13	a 1 value allows the reader to return the backscatter phase (start and end) of a Gen2 tag.
Bit 14	a 1 value allows the reader to return the backscatter frequency of a Gen2 tag in addition to the ID

InventoryTag Method (Int16, Byte[], Int16, Int16)

Description:

A call to this method will execute a read cycle on each read point linked to the logical source.

Parameters:

Name	Description
bank	A value representing the memory bank where apply the filter.
Mask	A byte array representing the bitmask to apply.
MaskLength	A value representing the bit-oriented length of the bitmask.
Position	A value representing the first bit of ID where the match will start.

Return value:

An array containing the CAENRFIDTag objects representing the tags read from the read points.

Syntax:

C# representation:

```
public CAENRFIDTag[]        InventoryTag(
                            short               bank,
                            byte[]              Mask,
                            short               MaskLength,
                            short               Position)
```

Java and Android representation:

```
public CAENRFIDTag[]        InventoryTag(
                            short               bank,
                            byte[]              Mask,
                            short               MaskLength,
                            short               Position)
                            throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_BankFilteredInventoryTag (
                            CAENRFIDHandle      handle,
                            char                *SourceName,
                            short               bank,
                            short               Position,
                            short               MaskLength,
                            char                *Mask,
                            CAENRFIDTag         **Receive,
                            int                 *Size);
```

Swift representation:

```
func inventoryTag(
                            bank: CAENRFIDTag.MemBanks = .epc,
                            mask: [UInt8] = [],
                            maskLength: UInt16 = 0,
                            position: UInt16 = 0
                            ) throws -> [CAENRFIDTag]?
```

Remarks:

Depending on the air protocol setting it will execute the appropriate anticollision algorithm. This version of the method permits to specify a bitmask for filtering tag populations as described by the EPC Class1 Gen2 (ISO18000-6C) air protocol. The filtering will be performed on the memory bank specified by bank parameter, starting at the bit indicated by the Position index and for a MaskLength length. The method will return only the tags that match the given Mask. Passing a zero value for MaskLength it performs as the non-filtering InventoryTag method.

InventoryTag Method (Int16, Byte[], Int16, Int16, Int16)

Description:

A call to this method will execute a read cycle on each read point linked to the logical source.

Parameters:

Name	Description
bank	A value representing the memory bank where apply the filter.
Mask	A byte array representing the bitmask to apply.
MaskLength	A value representing the bit-oriented length of the bitmask.
Position	A value representing the first bit of ID where the match will start.
Flag	A bitmask representing the InventoryTag options.

Return value:

An array containing the CAENRFIDTag objects representing the tags read from the read points.

Syntax:

C# representation:

```
public CAENRFIDTag[]        InventoryTag(
                            short               bank,
                            byte[]              Mask,
                            short               MaskLength,
                            short               Position,
                            short               Flag)
```

Java and Android representation:

```
public CAENRFIDTag[]        InventoryTag(
                            short               bank,
                            byte[]              Mask,
                            short               MaskLength,
                            short               Position,
                            short               Flag)
                            throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_BankFilteredFlagInventoryTag (
                            CAENRFIDHandle      handle,
                            char                *SourceName,
                            short               bank,
                            short               Position,
                            short               MaskLength,
                            char                *Mask,
                            unsigned char       Flag,
                            CAENRFIDTag         **Receive,
                            int                 *Size);
```

Swift representation:

```
func inventoryTag(
                            bank: CAENRFIDTag.MemBanks = .epc,
                            mask: [UInt8] = [],
                            maskLength: UInt16 = 0,
                            position: UInt16 = 0,
                            flag: UInt16 = 0
                            ) throws -> [CAENRFIDTag]?
```

Remarks:

Depending on the air protocol setting it will execute the appropriate anticollision algorithm. This version of the method permits to specify a bitmask for filtering tag populations as described by the EPC Class1 Gen2 (ISO18000-6C) air protocol. The filtering will be performed on the memory bank specified by bank parameter, starting at the bit indicated by the Position index and for a MaskLength length. The method will return only the tags that match the given Mask. Passing a zero value for MaskLength it performs as the non-filtering InventoryTag method. The Flags parameter permits to set InventoryTag method's options. In this case bit 1 and 2 of the flag (continuous and framed mode) are ignored.

Flag value meaning	
Bit 0	RSSI: a 1 value indicates that the reader will transmit the RSSI (Return Signal Strength Indicator) in the response.
Bit 3	Compact data: a 1 value indicates that only the EPC of the tag will be returned by the reader, a 0 value indicates that the complete data will be returned. In case that the compact option is enabled all the other data will be populated by this library with fakes values.
Bit 4	TID reading: a 1 value indicates that also the TID of the tag will be returned by the reader together with the other information.
Bit 5	Event trigger: when this flag is set together with the continuous acquisition flag, the inventory cycle is performed in the same way of the continuous mode with the only difference that the inventory command is sent only by pressing the button (only for readers equipped with button).
Bit 6	XPC: a 1 value allows the reader to get the XPC word if backscattered by a tag.Tags that do not backscatter the XPC words will return an XPC array with all the 4 bytes set to 0
Bit 8	PC: a 1 value allows the reader to return the PC of a Gen2 tag in addition to the ID.
Bit 13	a 1 value allows the reader to return the backscatter phase (start and end) of a Gen2 tag.
Bit 14	a 1 value allows the reader to return the backscatter frequency of a Gen2 tag in addition to the ID

FreeTagsMemory

Description:

The function permits to free the allocated memory by CAENRFID_InventoryTag.

Unlike the C#/Java languages where objects are automatically destroyed by the Runtime Environment, in C language it is necessary to explicitly deallocate the memory allocated by the identified tags. To do that, the FreeTagsMemory function is available, passing the pointer to the identified tags list.

Parameters:

Name	Description
Tags	tags array returned by one of the inventory family function.

Syntax:

C# representation:

Not supported

Java and Android representation:

Not supported

C representation:

```
void         CAENRFID_FreeTagsMemory(
                                    CAENRFIDTag           **Tags);
```

Swift representation:

Not supported

isReadPointPresent Method

Description:

This method checks if a read point is present in the logical source.

Parameters:

Name	Description
ReadPoint	A string representing the name of the read point (antenna).

Return value:

A boolean value representing the presence of a read point in the logical source (true means that it is present, false if it is not present).

Syntax:

C# representation:

```
public bool          isReadPointPresent(
                            string                  ReadPoint)
```

Java and Android representation:

```
public boolean       isReadPointPresent(
                            java.lang.String        ReadPoint)
                            throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes   CAENRFID_isReadPointPresent(
                            CAENRFIDHandle          handle,
                            char                    *ReadPoint,
                            char                    *SourceName,
                            short                   *isPresent);
```

Swift representation:

```
func isReadPointPresent(_ readPoint: String) throws -> Bool
```

KillTag_EPC_C1G2 Method

KillTag_EPC_C1G2 Method (CAENRFIDTag, Int32)

Description:

This method can be used to kill a EPC Class 1 Gen 2 tag.

Parameters:

Name	Description
Tag	The CAENRFIDTag representing the tag to be killed.
Password	The tag kill password.

Syntax:

C# representation:

```
public void          KillTag_EPC_C1G2(
                            CAENRFIDTag             Tag,
                            int                     Password)
```

Java and Android representation:

```
public void          KillTag_EPC_C1G2(
                            CAENRFIDTag             Tag,
                            int                     Password)
                            throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes   CAENRFID_KillTag_EPC_C1G2(
                            CAENRFIDHandle          handle,
                            CAENRFIDTag             *Tag,
                            int                     Password);
```

Swift representation:

```
func killTag_EPC_C1G2(
                            tag: CAENRFIDTag,
                            killPassword: UInt32
                            ) throws
```

KillTag_EPC_C1G2 Method (Int16, Int16, Int16, Byte[], Int32)

Description:

This method can be used to kill a EPC Class 1 Gen 2 tag.

Parameters:

Name	Description
BankMask	Memory bank for tag identification.
PositionMask	Bit position (from the start of the selected bank) where apply the mask to match.
LengthMask	Length of the mask.
Mask	Mask of byte.
Password	The tag kill password.

Syntax:

C# representation:

```
public void         KillTag_EPC_C1G2(
                            short               BankMask,
                            short               PositionMask,
                            short               LengthMask,
                            byte[]              Mask,
                            int                 Password)
```

Java and Android representation:

```
public void         KillTag_EPC_C1G2(
                            short               BankMask,
                            short               PositionMask,
                            short               LengthMask,
                            byte[]              Mask,
                            int                 Password)
                            throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_BankFilteredKillTag_EPC_C1G2(
                            CAENRFIDHandle      handle,
                            char                *SourceName,
                            short               BankMask,
                            short               PositionMask,
                            short               LengthMask,
                            char                *Mask,
                            int                 Password);
```

Swift representation:

```
func killTag_EPC_C1G2(
                            maskBank: CAENRFIDTag.MemBanks,
                            maskPosition: UInt16,
                            maskLength: UInt16,
                            mask: [UInt8],
                            killPassword: UInt32
                            ) throws
```

LockBlockPermaLock_EPC_C1G2 Method

Warning: This method is not supported by all readers. For more information, please contact the support at *support@caenrfid.com*.

Description:

This method implements the BLockPermaLock with ReadLock=1 as specified in EPC C1G2 rev. 1.2.0 protocol.

Parameters:

Name	Description
Tag	The CAENRFIDTag representing the tag to be written.
MemBank	The memory bank where to write the data.
BlockPtr	The address where to start writing the data.
BlockRange	The number of word of the mask.
Mask	A bitmask that permit to select which of the four bytes have to be locked (i.e. mask 0x05 write the bytes on position Address + 1 and Address + 3).
AccessPassword	The access password.

Syntax:

C# representation:

```
public void         LockBlockPermaLock_EPC_C1G2(
                            CAENRFIDTag             Tag,
                            short                   MemBank,
                            short                   BlockPtr,
                            short                   BlockRange,
                            byte[]                  Mask,
                            int                     AccessPassword)
```

Java and Android representation:

```
public void         LockBlockPermaLock_EPC_C1G2(
                            CAENRFIDTag             Tag,
                            short                   MemBank,
                            short                   BlockPtr,
                            short                   BlockRange,
                            byte[]                  Mask,
                            int                     AccessPassword)
                            throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_LockBlockPermaLock_EPC_C1G2(
                            CAENRFIDHandle          handle,
                            CAENRFIDTag             *Tag,
                            short                   MemBank,
                            short                   BlockPtr,
                            short                   BlockRange,
                            byte[]                  Mask,
                            int                     AccessPassword);
```

Swift representation:

```
func lockBlockPermaLock_EPC_C1G2(
                            tag: CAENRFIDTag,
                            bank: CAENRFIDTag.MemBanks,
                            blockPtr: UInt16,
                            blockRange: UInt16,
                            mask: [UInt8],
                            accessPassword: UInt32
                            ) throws
```

LockTag_EPC_C1G2 Method

LockTag_EPC_C1G2 Method (CAENRFIDTag, Int32)

Description:

This method can be used to lock a memory bank of a EPC Class 1 Gen 2 tag.

Parameters:

Name	Description
Tag	The CAENRFIDTag representing the tag to be locked.
Payload	The Payload parameter for the lock command as defined by the EPC Class 1 Gen 2 protocol specification.

Syntax:

C# representation:

```
public void         LockTag_EPC_C1G2(
                            CAENRFIDTag           Tag,
                            int                   Payload)
```

Java and Android representation:

```
public void         LockTag_EPC_C1G2(
                            CAENRFIDTag           Tag,
                            int                   Payload)
                            throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_LockTag_EPC_C1G2(
                            CAENRFIDHandle        handle,
                            CAENRFIDTag           *Tag,
                            int                   Payload);
```

Swift representation:

```
func lockTag_EPC_C1G2(
                            tag: CAENRFIDTag,
                            payload: UInt32
                            ) throws
```

LockTag_EPC_C1G2 Method (CAENRFIDTag, Int32, Int32)

Description:

This method can be used to lock a memory bank of a EPC Class 1 Gen 2 tag after having put it in Secured state using the Access command.

Parameters:

Name	Description
Tag	The CAENRFIDTag representing the tag to be locked.
Payload	The Payload parameter for the lock command as defined by the EPC Class 1 Gen 2 protocol specification.
AccessPassword	The access password.

Syntax:

C# representation:

```
public void         LockTag_EPC_C1G2(
                            CAENRFIDTag           Tag,
                            int                   Payload,
                            int                   AccessPassword)
```

Java and Android representation:

```
public void         LockTag_EPC_C1G2(
                            CAENRFIDTag           Tag,
                            int                   Payload,
                            int                   AccessPassword)
                            throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_SecureLockTag_EPC_C1G2(
                            CAENRFIDHandle        handle,
                            CAENRFIDTag           *Tag,
                            int                   Payload,
                            int                   AccessPassword);
```

Swift representation:

```
func lockTag_EPC_C1G2(
                            tag: CAENRFIDTag,
                            payload: UInt32,
                            accessPassword: UInt32
                            ) throws
```

LockTag_EPC_C1G2 Method (Int16, Int16, Int16, Byte[], Int32)

Description:

This method can be used to lock a memory bank of a EPC Class 1 Gen 2 tag.

Parameters:

Name	Description
BankMask	Memory bank for tag identification.
PositionMask	Bit position (from the start of the selected bank) where apply the mask to match.
LengthMask	Length of the mask.
Mask	Mask of byte.
Payload	The Payload parameter for the lock command as defined by the EPC Class 1 Gen 2 protocol specification.

Syntax:

C# representation:

```
public void         LockTag_EPC_C1G2(
                            short                 BankMask,
                            short                 PositionMask,
                            short                 LengthMask,
                            byte[]                Mask,
                            int                   Payload)
```

Java and Android representation:

```
public void  LockTag_EPC_C1G2(
                            short                 BankMask,
                            short                 PositionMask,
                            short                 LengthMask,
                            byte[]                Mask,
                            int                   Payload)
                            throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_BankFilteredLockTag_EPC_C1G2(
                            CAENRFIDHandle        handle,
                            char                  *SourceName,
                            short                 BankMask,
                            short                 PositionMask,
                            short                 LengthMask,
                            char                  *Mask,
                            int                   Payload);
```

Swift representation:

```
func lockTag_EPC_C1G2(
                            bankMask: CAENRFIDTag.MemBanks,
                            maskPosition: UInt16,
                            maskLength: UInt16,
                            mask: [UInt8],
                            payload: UInt32) throws
```

LockTag_EPC_C1G2 Method (Int16, Int16, Int16, Byte[], Int32, Int32)

Description:

This method can be used to lock a memory bank of a EPC Class 1 Gen 2 tag after having put it in Secured state using the Access command.

Parameters:

Name	Description
BankMask	Memory bank for tag identification.
PositionMask	Bit position (from the start of the selected bank) where apply the mask to match.
LengthMask	Length of the mask.
Mask	Mask of byte.
Payload	The Payload parameter for the lock command as defined by the EPC Class 1 Gen 2 protocol specification.
AccessPassword	Access password.

Syntax:

C# representation:

```
public void         LockTag_EPC_C1G2(
                            short                   BankMask,
                            short                   PositionMask,
                            short                   LengthMask,
                            byte[]                  Mask,
                            int                     Payload,
                            int                     AccessPassword)
```

Java and Android representation:

```
public void         LockTag_EPC_C1G2(
                            short                   BankMask,
                            short                   PositionMask,
                            short                   LengthMask,
                            byte[]                  Mask,
                            int                     Payload,
                            int                     AccessPassword)
                            throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_SecureBankFilteredLockTag_EPC_C1G2(
                            CAENRFIDHandle          handle,
                            char                    *SourceName,
                            short                   BankMask,
                            short                   PositionMask,
                            short                   LengthMask,
                            char                    *Mask,
                            int                     Payload,
                            int                     AccessPassword);
```

Swift representation:

```
func lockTag_EPC_C1G2(
                            bankMask: CAENRFIDTag.MemBanks,
                            maskPosition: UInt16,
                            maskLength: UInt16,
                            mask: [UInt8],
                            payload: UInt32,
                            accessPassword: UInt32) throws
```

ProgramID_EPC_C1G2 Method

ProgramID_EPC_C1G2 Method (CAENRFIDTag, Int16)

Description:

This method can be used to write the EPC of a EPC Class 1 Gen 2 tag.

Parameters:

Name	Description
Tag	The CAENRFIDTag representing the tag to be programmed, the ID contained in this object will be programmed into the tag.
NSI	The Numbering System Identifier as defined in EPC Class 1 Gen 2 protocol specifications.

Syntax:

C# representation:

```
public void         ProgramID_EPC_C1G2(
                            CAENRFIDTag           Tag,
                            short                 NSI)
```

Java and Android representation:

```
public void         ProgramID_EPC_C1G2(
                            CAENRFIDTag           Tag,
                            short                 NSI)
                            throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_ProgramID_EPC_C1G2(
                            CAENRFIDHandle        handle,
                            CAENRFIDTag           *Tag,
                            unsigned short        NSI);
```

Swift representation:

```
func programID_EPC_C1G2(
                            tag: CAENRFIDTag,
                            nsi: UInt16 = 0
                            ) throws
```

ProgramID_EPC_C1G2 Method (CAENRFIDTag, Int16, Int32)

Description:

This method can be used to write the EPC of a EPC Class 1 Gen 2 tag after having put it in Secured state using the Access command.

Parameters:

Name	Description
Tag	The CAENRFIDTag representing the tag to be programmed, the ID contained in this object will be programmed into the tag.
NSI	The Numbering System Identifier as defined in EPC Class 1 Gen 2 protocol specifications.
AccessPassword	The access password.

Syntax:

C# representation:

```
public void         ProgramID_EPC_C1G2(
                    CAENRFIDTag           Tag,
                    short                 NSI,
                    int                   AccessPassword)
```

Java and Android representation:

```
public void         ProgramID_EPC_C1G2(
                    CAENRFIDTag           Tag,
                    short                 NSI,
                    int                   AccessPassword)
                    throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_SecureProgramID_EPC_C1G2(
                    CAENRFIDHandle        handle,
                    CAENRFIDTag           *Tag,
                    unsigned short        NSI,
                    int                   AccessPassword);
```

Swift representation:

```
func programID_EPC_C1G2(
                    tag: CAENRFIDTag,
                    nsi: UInt16 = 0,
                    accessPassword : UInt32
                    ) throws
```

ReadBLockPermalock_EPC_C1G2 Method

Warning: This method is not supported by all readers. For more information, please contact the support at *support@caenrfid.com*.

Description:

This method implements the BLockPermaLock with ReadLock=0 as specified in EPCC1G2 rev. 1.2.0 protocol.

Parameters:

Name	Description
Tag	The CAENRFIDTag representing the tag to be read.
MemBank	The memory bank where to read the data.
Blockptr	The address where to start reading the data.
BlockRange	The number of word to be read.
AccessPassword	The access password.

Return value:

An array of bytes representing the data read from the tag.

Syntax:

C# representation:

```
public byte[]       ReadBLockPermalock_EPC_C1G2(
                           CAENRFIDTag           Tag,
                           short                 MemBank,
                           short                 Blockptr,
                           short                 BlockRange,
                           int                   AccessPassword)
```

Java and Android representation:

```
public byte[]       ReadBLockPermalock_EPC_C1G2(
                           CAENRFIDTag           Tag,
                           short                 MemBank,
                           short                 Blockptr,
                           short                 BlockRange,
                           int                   AccessPassword)
                           throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_ReadBLockPermalock_EPC_C1G2(
                           CAENRFIDHandle        handle,
                           CAENRFIDTag           *Tag,
                           short                 MemBank,
                           short                 Blockptr,
                           short                 BlockRange,
                           int                   AccessPassword)
```

Swift representation:

```
func readBlockLockPermalock_EPC_C1G2(
                            tag: CAENRFIDTag,
                            bank: CAENRFIDTag.MemBanks,
                            blockPtr: UInt16,
                            blockRange: UInt16,
                            accessPassword: UInt32
                            ) throws -> [UInt8]?
```

ReadTagData_EPC_C1G2 Method

ReadTagData_EPC_C1G2 Method (CAENRFIDTag, Int16, Int16, Int16)

Description:

This method can be used to read a portion of memory in an ISO18000-6C (EPC Class1 Gen2) tag.

Parameters:

Name	Description
Tag	The CAENRFIDTag representing the tag to be read.
MemBank	The memory bank where to read the data.
Address	The address where to start reading the data.
Length	The number of byte to be read.

Return value:

An array of bytes representing the data read from the tag.

Syntax:

C# representation:

```
public byte[]       ReadTagData_EPC_C1G2(
                            CAENRFIDTag             Tag,
                            short                   MemBank,
                            short                   Address,
                            short                   Length)
```

Java and Android representation:

```
public byte[]       ReadTagData_EPC_C1G2(
                            CAENRFIDTag             Tag,
                            short                   MemBank,
                            short                   Address,
                            short                   Length)
                            throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_ReadTagData_EPC_C1G2(
                            CAENRFIDHandle          handle,
                            CAENRFIDTag             *Tag,
                            short                   MemBank,
                            int                     Address,
                            int                     Length,
                            void                    *Data);
```

Swift representation:

```
func readTagData_EPC_C1G2(
                            tag: CAENRFIDTag,
                            memBank: CAENRFIDTag.MemBanks,
                            address: UInt16,
                            length: UInt16
                            ) throws -> [UInt8]?
```

ReadTagData_EPC_C1G2 Method (CAENRFIDTag, Int16, Int16, Int16, Int32)

Description:

This method can be used to read a portion of memory in an ISO18000-6C (EPC Class1 Gen2) tag after having put the tag in Secured state using the Access command.

Parameters:

Name	Description
Tag	The CAENRFIDTag representing the tag to be read.
MemBank	The memory bank where to read the data.
Address	The address where to start reading the data.
Length	The number of byte to be read.
AccessPassword	The access password.

Return value:

An array of bytes representing the data read from the tag.

Syntax:

C# representation:

```
public byte[]       ReadTagData_EPC_C1G2(
                            CAENRFIDTag           Tag,
                            short                 MemBank,
                            short                 Address,
                            short                 Length,
                            int                   AccessPassword)
```

Java and Android representation:

```
public byte[]       ReadTagData_EPC_C1G2(
                            CAENRFIDTag           Tag,
                            short                 MemBank,
                            short                 Address,
                            short                 Length,
                            int                   AccessPassword)
                            throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_SecureReadTagData_EPC_C1G2 (
                            CAENRFIDHandle        handle,
                            CAENRFIDTag           *Tag,
                            short                 MemBank,
                            int                   Address,
                            int                   Length,
                            int                   AccessPassword,
                            void                  *Data);
```

Swift representation:

```
func readTagData_EPC_C1G2(
                            tag: CAENRFIDTag,
                            memBank: CAENRFIDTag.MemBanks,
                            address: UInt16,
                            length: UInt16,
                            accessPassword: UInt32
                            ) throws -> [UInt8]?
```

ReadTagData_EPC_C1G2 Method (Int16, Int16, Int16, Byte[], Int16, Int16, Int16)

Description:

This method can be used to read a portion of memory in an ISO18000-6C (EPC Class1 Gen2) tag. In this case the target tag is identified by 'LenghtMask' bytes of passed mask placed in a memory bank 'BankMask' at 'PositionMask' byte from bank starting address byte.

Parameters:

Name	Description
BankMask	Memory bank for tag identificantion.
PositionMask	Bit position (from the start of the selected bank) where apply the mask to match.
LengthMask	Length of the mask.
Mask	Mask of byte.
MemBank	Memory bank where read.
Address	Address where starts reading.
Length	Number of byte to read.

Return value:

An array of bytes representing the data read from the tag.

Syntax:

C# representation:

```
public byte[]       ReadTagData_EPC_C1G2(
                            short                   BankMask,
                            short                   PositionMask,
                            short                   LengthMask,
                            byte[]                  Mask,
                            short                   MemBank,
                            short                   Address,
                            short                   Length)
```

Java and Android representation:

```
public byte[]       ReadTagData_EPC_C1G2(
                            short                   BankMask,
                            short                   PositionMask,
                            short                   LengthMask,
                            byte[]                  Mask,
                            short                   MemBank,
                            short                   Address,
                            short                   Length)
                            throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_BankFilteredReadTagData_EPC_C1G2(
                            CAENRFIDHandle          handle,
                            char                    *SourceName,
                            short                   BankMask,
                            short                   PositionMask,
                            short                   LengthMask,
                            char                    *Mask,
                            short                   MemBank,
                            int                     Address,
                            int                     Length,
                            void                    *Data);
```

Swift representation:

```
func readTagData_EPC_C1G2(
                            bankMask: CAENRFIDTag.MemBanks,
                            positionMask: UInt16,
                            lengthMask: UInt16,
                            maskId: [UInt8],
                            memBank: CAENRFIDTag.MemBanks,
                            address: UInt16,
                            length: UInt16
                            ) throws -> [UInt8]?
```

ReadTagData_EPC_C1G2 Method (Int16, Int16, Int16, Byte[], Int16, Int16, Int16, Int32)

Description:

This method can be used to read a portion of memory in an ISO18000-6C (EPC Class1 Gen2) tag. In this case the target tag is identified by 'LenghtMask' bytes of passed mask placed in a memory bank 'BankMask' at 'PositionMask' byte from bank starting address byte. This is the secure version using the Access command.

Parameters:

Name	Description
BankMask	Memory bank for tag identificantion.
PositionMask	Bit position (from the start of the selected bank) where apply the mask to match.
LengthMask	Length of the mask.
Mask	Mask of byte.
MemBank	Memory bank where read.
Address	Address where starts reading.
Length	Number of byte to read.
AccessPassword	Access Password.

Return value:

An array of bytes representing the data read from the tag.

Syntax:

C# representation:

```
public byte[]       ReadTagData_EPC_C1G2(
                            short                   BankMask,
                            short                   PositionMask,
                            short                   LengthMask,
                            byte[]                  Mask,
                            short                   MemBank,
                            short                   Address,
                            short                   Length,
                            int                     AccessPassword)
```

Java and Android representation:

```
public byte[]       ReadTagData_EPC_C1G2(
                            short                   BankMask,
                            short                   PositionMask,
                            short                   LengthMask,
                            byte[]                  Mask,
                            short                   MemBank,
                            short                   Address,
                            short                   Length,
                            int                     AccessPassword)
                            throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_SecureBankFilteredReadTagData_EPC_C1G2 (
                            CAENRFIDHandle          handle,
                            char                    *SourceName,
                            short                   BankMask,
                            short                   PositionMask,
                            short                   LengthMask,
                            byte[]                  Mask,
                            short                   MemBank,
                            int                     Address,
                            int                     Length,
                            void                    *Data,
                            int                     AccessPassword);
```

Swift representation:

```
func readTagData_EPC_C1G2(
                            bankMask: CAENRFIDTag.MemBanks,
                            positionMask: UInt16,
                            lengthMask: UInt16,
                            maskId: [UInt8],
                            memBank: CAENRFIDTag.MemBanks,
                            address: UInt16,
                            length: UInt16,
                            accessPassword: UInt32
                            ) throws -> [UInt8]?
```

RemoveReadPoint Method

Description:

This method removes a read point from the logical source.

Parameters:

Name	Description
ReadPoint	A string representing the name of the read point (antenna).

Syntax:

C# representation:

```
public void         RemoveReadPoint(
                            string              ReadPoint)
```

Java and Android representation:

```
public void         RemoveReadPoint(
                            java.lang.String    ReadPoint)
                            throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_RemoveReadPoint(
                            CAENRFIDHandle      handle,
                            char                *SourceName,
                            char                *ReadPoint);
```

Swift representation:

```
func removeReadPoint(_ readPoint: String) throws
```

SetInventoryCounts Method

Description:

This method can be used to set the current setting for the number of inventory counts performed by the logical source after pressing the TRIGGER during the inventory algorithm execution.

Parameters:

Name	Description
Value	The number of read cycles.

Syntax:

C# representation:

```
public void             SetInventoryCounts(
                                int                 value);
```

Java and Android representation:

```
public void             SetInventoryCounts(
                                int                 value)
                        throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes      CAENRFID_SetInventoryCounts(
                                CAENRFIDHandle      handle,
                                char                *SourceName,
                                int                 *value);
```

Swift representation:

```
func setInventoryCounts(_ count: UInt) throws
```

SetInventoryDwellTime Method

Description:

This method can be used to sets the inventory execution time (msec) used by the logical source during the inventory algorithm execution.

Parameters:

Name	Description
Value	The execution time in ms

Syntax:

C# representation:

```
public int              SetInventoryDwellTime(
                               int                  value);
```

Java and Android representation:

```
public int              SetInventoryDwellTime(
                               int                  value)
                        throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes      CAENRFID_SetInventoryDwellTime(
                               CAENRFIDHandle       handle,
                               char                 *SourceName,
                               int                  value);
```

Swift representation:

```
func setInventoryDwellTime(_ dwellTime: UInt) throws
```

SetInventoryQuietTime Method

Description:

This method can be used to sets the inventory quiet time (msec) used by the logical source during the inventory algorithm execution.

Parameters:

Name	Description
Value	The quiet time in ms

Syntax:

C# representation:

```
public int              SetInventoryQuietTime(
                               int                  value);
```

Java and Android representation:

```
public int              SetInventoryQuietTime(
                               int                  value)
                        throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes      CAENRFID_SetInventoryQuietTime(
                               CAENRFIDHandle       handle,
                               char                 *SourceName,
                               int                  value);
```

Swift representation:

```
func setInventoryQuietTime(_ quietTime: UInt) throws
```

SetQ_EPC_C1G2 Method

Description:

This method can be used to set the initial Q value (see EPC Class1 Gen2 protocol specification) used by the anticollision algorithm when called on this logical source.

Parameters:

Name	Description
Value	The initial Q value setting.

Syntax:

C# representation:

```
public void         SetQ_EPC_C1G2(
                            int                     Value)
```

Java and Android representation:

```
public void         SetQ_EPC_C1G2(
                            int                     Value)
                            throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_SetQValue_EPC_C1G2(
                            CAENRFIDHandle          handle,
                            char                    *SourceName,
                            int                     Value);
```

Swift representation:

```
func setQ_EPC_C1G2(_ qValue: UInt) throws
```

SetMaxQ_C1G2 Method

Description:

This method can be used to set the maximum Q value (see EPC Class1 Gen2 protocol specification) used by the anticollision algorithm when called on this logical source.

Parameters:

Name	Description
Value	The maximum Q value setting.

Syntax:

C# representation:

```
public void         SetMaxQ_EPC_C1G2(
                    int                     Value)
```

Java and Android representation:

```
public void         SetMaxQ_EPC_C1G2(
                    int                     Value)
                    throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_SetMaxQValue_EPC_C1G2(
                    CAENRFIDHandle          handle,
                    char                    *SourceName,
                    int                     Value);
```

Swift representation:

```
func setMaxQ_EPC_C1G2(_ qValue: UInt) throws
```

SetMinQ_EPC_C1G2 Method

Description:

This method can be used to set the minimum Q value (see EPC Class1 Gen2 protocol specification) used by the anticollision algorithm when called on this logical source.

Parameters:

Name	Description
Value	The minimum Q value setting.

Syntax:

C# representation:

```
public void         SetMinQ_EPC_C1G2(
                    int                 Value)
```

Java and Android representation:

```
public void         SetMinQ_EPC_C1G2(
                    int                 Value)
                    throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_SetMinQValue_EPC_C1G2(
                    CAENRFIDHandle      handle,
                    char                *SourceName,
                    int                 Value);
```

Swift representation:

```
func setMinQ_EPC_C1G2(_ qValue: UInt) throws
```

SetNumMinQ_EPC_C1G2 Method

Description:

This method can be used to set the number of inventory round count to perform with minimum EPC Class 1 Gen. 2 Q value and no tag has been found on each readpoint of this logical source, before stop the inventory execution.

Parameters:

Name	Description
Value	The max times of inventory execution with minimum Q value setting.

Syntax:

C# representation:

```
public void         SetNumMinQ_EPC_C1G2(
                           int                 Value)
```

Java and Android representation:

```
public void         SetNumMinQ_EPC_C1G2(
                           int                 Value)
                           throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_SetNumMinQValue_EPC_C1G2(
                           CAENRFIDHandle      handle,
                           char                *SourceName,
                           int                 Value);
```

Swift representation:

```
func setNumMinQ_EPC_C1G2(_ qValue: UInt) throws
```

SetReadCycle Method

Description:

This method sets the number of read cycles to be performed by the logical source during the inventory algorithm execution.

Parameters:

Name	Description
value	The number of read cycles.

Syntax:

C# representation:

```
public void         SetReadCycle(
                            int                     value)
```

Java and Android representation:

```
public void         SetReadCycle(
                            int                     value)
                            throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_SetReadCycle(
                            CAENRFIDHandle          handle,
                            char                    *SourceName,
                            int                     value);
```

Swift representation:

```
func setReadCycle(_ cycleNum: UInt) throws
```

SetSelected_EPC_C1G2 Method

Description:

This method can be used to set the Selected flag (see EPC Class1 Gen2 protocol specification) used by the anticollision algorithm when called on this logical source.

Parameters:

Name	Description
Value	The Selected flag value.

Syntax:

C# representation:

```
public void         SetSelected_EPC_C1G2(
                            CAENRFIDLogicalSourceConstants      Value)
```

Java and Android representation:

```
public void         SetSelected_EPC_C1G2(
                            CAENRFIDLogicalSourceConstants      Value)
                            throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_SetSelected_EPC_C1G2(
                            CAENRFIDHandle                      handle,
                            char                                *SourceName,
                            CAENRFIDLogicalSourceConstants      Value);
```

Swift representation:

```
func setSelected_EPC_C1G2(selected: CAENRFIDLogicalSourceConstants) throws
```

SetSession_EPC_C1G2 Method

Description:

This method can be used to set the Session (see EPC Class1 Gen2 protocol specification) used by the anticollision algorithm when called on this logical source.

Parameters:

Name	Description
Value	The Session value.

Syntax:

C# representation:

```
public void         SetSession_EPC_C1G2(
                        CAENRFIDLogicalSourceConstants      Value)
```

Java and Android representation:

```
public void         SetSession_EPC_C1G2(
                        CAENRFIDLogicalSourceConstants      Value)
                        throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_SetSession_EPC_C1G2(
                        CAENRFIDHandle                      handle,
                        char                                *SourceName,
                        CAENRFIDLogicalSourceConstants      Value);
```

Swift representation:

```
func setSession_EPC_C1G2(session: CAENRFIDLogicalSourceConstants) throws
```

SetTarget_EPC_C1G2 Method

Description:

This method can be used to set the Target setting (see EPC Class1 Gen2 protocol specification) used by the anticollision algorithm when called on this logical source.

Parameters:

Name	Description
Value	The Target value.

Syntax:

C# representation:

```
public void         SetTarget_EPC_C1G2(
                        CAENRFIDLogicalSourceConstants      Value)
```

Java and Android representation:

```
public void         SetTarget_EPC_C1G2(
                        CAENRFIDLogicalSourceConstants      Value)
                        throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_SetTarget_EPC_C1G2(
                        CAENRFIDHandle                      handle,
                        char                                *SourceName,
                        CAENRFIDLogicalSourceConstants      Value);
```

Swift representation:

```
func setTarget_EPC_C1G2(target: CAENRFIDLogicalSourceConstants) throws
```

SetTIDLength Method

Description:

This method can be used to set the excepted TID length when TID reading is enabled as flag in inventory methods.

Parameters:

Name	Description
Value	The expected TID length expressed in bytes.

Syntax:

C# representation:

```
public void         SetTIDLength_EPC_C1G2(
                        int     Value)
```

Java and Android representation:

```
public void         SetTIDLength_EPC_C1G2(
                        int     Value)
                        throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_SetTIDLength_EPC_C1G2(
                        CAENRFIDHandle                      handle,
                        char                                *SourceName,
                        int     Value);
```

Swift representation:

```
func setTIDLength(tidLength: UInt) throws
```

Untraceable_EPC_C1G2 Method

Warning: This method is not supported by all readers. For more information, please contact the support at *support@caenrfid.com*.

Description:

This method allows an interrogator with an asserted Untraceable privilege to instruct a Tag to (a) alter the L and U bits in EPC memory,(b) hide memory from interrogators with a deasserted Untraceable privilege and/or (c) reduce its operating range for all interrogators.

Parameters:

Name	Description
Tag	The CAENRFIDTag representing the tag to be untraced
u	A boolean value for the U bit in XPC_W1 word.
hideEPC	Specify whether a Tag untraceably hides part of EPC memory
hideTID	Specify whether a Tag untraceably hides part of TID memory (Allowed values: 0,1,2)
hideUSER	Specify whether a Tag untraceably hides part of USER memory
range	A value specifying a tag operating range in terms of reading distance (allowed values: 0,1,2).
newEPCLen	A value specifying a new EPC length field
password	The access password

Syntax:

C# representation:

```
public void         Untraceable_EPC_C1G2(
                        CAENRFIDTag         Tag,
                        bool                u,
                        bool                hideEPC,
                        byte                hideTID,
                        bool                hideUser,
                        ushort              range,
                        ushort              newEPCLen,
                        uint                password )
```

Java and Android representation:

```
public void         Untraceable_EPC_C1G2(
                            CAENRFIDTag           Tag,
                            bool                  u,
                            bool                  hideEPC,
                            bool                  hideTID,
                            byte                  hideUser,
                            ushort                range,
                            ushort                newEPCLen,
                            uint                  password )
                            throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_Untraceable_EPC_C1G2(
                            CAENRFIDHandle        handle,
                            char*                 SourceName,
                            CAENRFIDTag           *Tag,
                            BOOL                  u,
                            BOOL                  hideEPC,
                            byte                  hideTID,
                            BOOL                  hideUser,
                            unsigned short        range,
                            unsigned short        newEPCLen,
                            int                   password);
```

Swift representation:

```
func untraceable_EPC_C1G2(
                           tag: CAENRFIDTag,
                           u: Bool,
                           hideEPC: Bool,
                           hideTID: UInt8,
                           hideUser: Bool,
                           range: UInt16,
                           newEPCLen: UInt16,
                           accessPassword: UInt32
                           ) throws
```

WriteTagData_EPC_C1G2 Method

WriteTagData_EPC_C1G2 Method (CAENRFIDTag, Int16, Int16, Int16, Byte[])

Description:

This method can be used to write a portion of memory in an ISO18000-6C (EPC Class1 Gen2) tag.

Parameters:

Name	Description
Tag	The CAENRFIDTag representing the tag to be written.
MemBank	The memory bank where to write the data.
Address	The address where to start writing the data.
Length	The number of byte to be written.
Data	An array of bytes representing the data to be written into the tag.

Syntax:

C# representation:

```
public void         WriteTagData_EPC_C1G2(
                            CAENRFIDTag           Tag,
                            short                 MemBank,
                            short                 Address,
                            short                 Length,
                            byte[]                Data)
```

Java and Android representation:

```
public void         WriteTagData_EPC_C1G2(
                            CAENRFIDTag           Tag,
                            short                 MemBank,
                            short                 Address,
                            short                 Length,
                            byte[]                Data)
                            throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_WriteTagData_EPC_C1G2(
                            CAENRFIDHandle        handle,
                            CAENRFIDTag           *Tag,
                            short                 MemBank,
                            int                   Address,
                            int                   Length,
                            void                  *Data);
```

Swift representation:

```
func writeTagData_EPC_C1G2(
                            tag: CAENRFIDTag,
                            bank: CAENRFIDTag.MemBanks,
                            address: UInt16,
                            length: UInt16,
                            data: [UInt8]
                            ) throws
```

WriteTagData_EPC_C1G2 Method (CAENRFIDTag, Int16, Int16, Int16, Byte[], Int32)

Description:

This method can be used to write a portion of memory in an ISO18000-6C (EPC Class1 Gen2) tag after having put the tag in Secured state using the Access command.

Parameters:

Name	Description
Tag	The CAENRFIDTag representing the tag to be written.
MemBank	The memory bank where to write the data.
Address	The address where to start writing the data.
Length	The number of byte to be written.
Data	An array of bytes representing the data to be written into the tag.
AccessPassword	The access password.

Syntax:

C# representation:

```
public void         WriteTagData_EPC_C1G2(
                            CAENRFIDTag           Tag,
                            short                 MemBank,
                            short                 Address,
                            short                 Length,
                            byte[]                Data,
                            int                   AccessPassword)
```

Java and Android representation:

```
public void         WriteTagData_EPC_C1G2(
                            CAENRFIDTag           Tag,
                            short                 MemBank,
                            short                 Address,
                            short                 Length,
                            byte[]                Data,
                            int                   AccessPassword)
                            throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_SecureWriteTagData_EPC_C1G2(
                            CAENRFIDHandle        handle,
                            CAENRFIDTag           *Tag,
                            short                 MemBank,
                            int                   Address,
                            int                   Length,
                            void                  *Data,
                            int                   AccessPassword);
```

Swift representation:

```
func writeTagData_EPC_C1G2(
                            tag: CAENRFIDTag,
                            bank: CAENRFIDTag.MemBanks,
                            address: UInt16,
                            length: UInt16,
                            data: [UInt8],
                            accessPassword: UInt32
                            ) throws
```

WriteTagData_EPC_C1G2 Method (Int16, Int16, Int16, Byte[], Int16, Int16, Int16, Byte[])

Description:

This method can be used to write a portion of memory in an ISO18000-6C (EPC Class1 Gen2) tag.

Parameters:

Name	Description
BankMask	Memory bank for tag identification.
PositionMask	Bit position (from the start of the selected bank) where apply the mask to match.
LengthMask	Length of the mask.
Mask	Mask of byte.
MemBank	The memory bank where to write the data.
Address	The address where to start writing the data.
Length	The number of byte to be written.
Data	An array of bytes representing the data to be written into the tag.

Syntax:

C# representation:

```
public void         WriteTagData_EPC_C1G2(
                            short               BankMask,
                            short               PositionMask,
                            short               LengthMask,
                            byte[]              Mask,
                            short               MemBank,
                            short               Address,
                            short               Length,
                            byte[]              Data)
```

Java and Android representation:

```
public void         WriteTagData_EPC_C1G2(
                            short               BankMask,
                            short               PositionMask,
                            short               LengthMask,
                            byte[]              Mask,
                            short               MemBank,
                            short               Address,
                            short               Length,
                            byte[]              Data)
                            throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_BankFilteredWriteTagData_EPC_C1G2(
                            CAENRFIDHandle      handle,
                            char                *SourceName,
                            short               BankMask,
                            short               PositionMask,
                            short               LengthMask,
                            char                *Mask,
                            short               MemBank,
                            int                 Address,
                            int                 Length,
                            void                *Data);
```

Swift representation:

```
func writeTagData_EPC_C1G2(
                            maskBank: CAENRFIDTag.MemBanks,
                            maskPosition: UInt16,
                            maskLength: UInt16,
                            maskId: [UInt8],
                            bank: CAENRFIDTag.MemBanks,
                            address: UInt16,
                            data: [UInt8],
                            length: UInt16) throws
```

WriteTagData_EPC_C1G2 Method (Int16, Int16, Int16, Byte[], Int16, Int16, Int16, Byte[], Int32)

Description:

This method can be used to write a portion of memory in an ISO18000-6C (EPC Class1 Gen2) tag after having put the tag in Secured state using the Access command.

Parameters:

Name	Description
BankMask	Memory bank for tag identification.
PositionMask	Bit position (from the start of the selected bank) where apply the mask to match.
LengthMask	Length of the mask.
Mask	Mask of byte.
MemBank	The memory bank where to write the data.
Address	The address where to start writing the data.
Length	The number of byte to be written.
Data	An array of bytes representing the data to be written into the tag.
AccessPassword	The access password.

Syntax:

C# representation:

```
public void         WriteTagData_EPC_C1G2(
                            short               BankMask,
                            short               PositionMask,
                            short               LengthMask,
                            byte[]              Mask,
                            short               MemBank,
                            short               Address,
                            short               Length,
                            byte[]              Data,
                            int                 AccessPassword)
```

Java and Android representation:

```
public void         WriteTagData_EPC_C1G2(
                            short               BankMask,
                            short               PositionMask,
                            short               LengthMask,
                            byte[]              Mask,
                            short               MemBank,
                            short               Address,
                            short               Length,
                            byte[]              Data,
                            int                 AccessPassword)
                            throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_SecureBankFilteredWriteTagData_EPC_C1G2(
                            CAENRFIDHandle        handle,
                            char                  *SourceName,
                            short                 BankMask,
                            short                 PositionMask,
                            short                 LengthMask,
                            char                  *Mask,
                            short                 MemBank,
                            int                   Address,
                            int                   Length,
                            void                  *Data,
                            int                   AccessPassword);
```

Swift representation:

```
func writeTagData_EPC_C1G2(
                            maskBank: CAENRFIDTag.MemBanks,
                            maskPosition: UInt16,
                            maskLength: UInt16,
                            maskId: [UInt8],
                            bank: CAENRFIDTag.MemBanks,
                            address: UInt16,
                            data: [UInt8],
                            length: UInt16,
                            accessPassword: UInt32) throws
```

CAENRFIDNotify Class

The CAENRFIDNotify class defines the structure of a notification message. Swift API use the same class either for synchronous and asynchronous inventory: CAENRFIDTag class (see CAENRFIDTag class).

In Java, Android, C# and Swift languages this class is composed by methods while in C language is present as a struct (for more information see *§ Overview on SDK* page 7):

C representation:

```
typedef struct {
                            byte                  ID[MAX_ID_LENGTH];
                            short                 Length;
                            char                  LogicalSource[MAX_LOGICAL_SOURCE_NAME];
                            char                  ReadPoint[MAX_READPOINT_NAME];
                            CAENRFIDProtocol      Type;
                            short                 RSSI;
                            byte                  TID[MAX TID SIZE];
                            short                 TIDLen;
                            byte                  XPC[XPC_LENGTH];
                            byte                  PC[PC_LENGTH];
                            float                 phaseBegin;
                            float                 phaseEnd;
                            long                  frequency;
                            CAENRFID_InventorySubCommandType subCmdCode;
                            CAENRFIDErrorCodes    subCmdResultCode;
                            short                 subCmdResultDataCount;
                            byte*                 subCmdResultData;
             } CAENRFIDNotify;
```

getDate Method

Description:

This method returns a timestamp representing the time at which the event was generated.

Return value:

The timestamp value.

Syntax:

C# representation:

```
public DateTime                    getDate()
```

Java and Android representation:

```
public java.util.Date              getDate()
```

C representation:

Not supported

Swift representation:

Not supported

getPC Method

Description:

This method represents the PC code in the tag.

Return value:

The tag Protocol Control code.

Syntax:

C# representation:

```
public byte[]                getPC()
```

Java and Android representation:

```
public byte[]                getPC()
```

C representation:

Not supported

Swift representation:

Not supported

getReadPoint Method

Description:

This method returns the read point that has detected the tag.

Return value:

The name of the read point that has detected the Tag.

Syntax:

C# representation:

```
public string                        getReadPoint()
```

Java and Android representation:

```
public java.lang.String              getReadPoint()
```

C representation:

Not supported

Swift representation:

Not supported

getRSSI Method

Description:

This method returns the RSSI value measured for the tag.

Return value:

The tag RSSI.

Syntax:

C# representation:

```
public short                         getRSSI()
```

Java and Android representation:

```
public short                         getRSSI()
```

C representation:

Not supported

Swift representation:

Not supported

getStatus Method

Description:

This method returns the event type associated to the tag.

Return value:

The event type associated to the Tag.

Syntax:

C# representation:

```
public CAENRFIDTagEventType        getStatus()
```

Java and Android representation:

```
public CAENRFIDTagEventType        getStatus()
```

C representation:

Not supported

Swift representation:

Not supported

getTagID Method

Description:

This method returns the tag ID (the EPC code in Gen2 tags).

Return value:

An array of bytes representing the tag ID (the EPC code in EPC Class 1 Gen 2 tags).

Syntax:

C# representation:

```
public byte[]                      getTagID()
```

Java and Android representation:

```
public byte[]                      getTagID()
```

C representation:

Not supported

Swift representation:

Not supported

getTagLength Method

Description:

This method returns the tag ID length.

Return value:

The tag length.

Syntax:

C# representation:

```
public short                       getTagLength()
```

Java and Android representation:

```
public short                       getTagLength()
```

C representation:

Not supported

Swift representation:

Not supported

getTagSource Method

Description:

This method returns the name of the logical source that has detected the tag.

Return value:

The name of the logical source that has detected the tag.

Syntax:

C# representation:

```
public string                   getTagSource()
```

Java and Android representation:

```
public java.lang.String         getTagSource()
```

C representation:

Not supported

Swift representation:

Not supported

getTagType Method

Description:

This method returns the air protocol of the tag.

Return value:

The air protocol of the tag.

Syntax:

C# representation:

```
public short                    getTagType()
```

Java and Android representation:

```
public CAENRFIDProtocol         getTagType()
```

C representation:

Not supported

Swift representation:

Not supported

getTID Method

Description:

This method returns the TID field value in a EPC Class 1 Gen 2 Tag

Return value:

The bytes of the TID field.

Syntax:

C# representation:

```
public byte[]                   getTID()
```

Java and Android representation:

```
public java.lang.String         getAntenna()
```

C representation:

Not supported

Swift representation:

Not supported

getXPC Method

Description:

This method returns the tag XPC words.

Return value:

The tag XPC words.

Syntax:

C# representation:

```
public byte[]               getXPC()
```

Java and Android representation:

```
public byte[]               getXPC()
```

C representation:

Not supported

Swift representation:

Not supported

getFrequency Method

Description:

Get the backscattering frequency.

Return value:

An integer value representing the frequency expressed in KHz.

Syntax:

C# representation:

```
public long         getFrequency()
```

Java and Android representation:

```
public long         getFrequency()
```

C representation:

Not supported

Swift representation:

Not supported

getPhaseBegin Method

Description:

Gets the backscattering phase at the begin of communication from reader to tag.

Return value:

A decimal value expressed in Degree.

Syntax:

C# representation:

```
public float        getPhaseBegin()
```

Java and Android representation:

```
public float        getPhaseBegin()
```

C representation:

Not supported

Swift representation:

Not supported

getPhaseEnd Method

Description:

Gets the backscattering phase at the end of communication from reader to tag.

Return value:

A decimal value expressed in Degree.

Syntax:

C# representation:

```
public float        getPhaseEnd()
```

Java and Android representation:

```
public float        getPhaseEnd()
```

C representation:

Not supported

Swift representation:

Not supported

CAENRFIDReader Class

The CAENRFIDReader class is used to create reader objects which permit to access to CAEN RFID readers' configuration and control commands.

Connect Method

Connect Method (CAENRFIDPort, string)

Description:

In C# and Java languages, this method starts the communication with the reader. It must be called before any other call to method of the CAENRFIDReader object. See § *Managing connections with the readers* page 8 for more information. For android Bluetooth connection see below § *Connect Method (BluetoothSocket)*

Parameters:

Name	Description
ConType	The communication link to use for the connection.
Address	Depending on ConType parameter: IP address for TCP/IP communications ("xxx.xxx.xxx.xxx"), COM port for RS232 communications ("COMx"), an index for USB communications (not yet supported). To specify a TCP port separate address and port by a semi-colon (ex: "192.168.0.1:2300").

Syntax:

C# representation:

```
public void         Connect(
                        CAENRFIDPort            ConType,
                        string                  Address)
```

Java and Android representation:

```
public void         Connect(
                        CAENRFIDPort          ConType,
                        java.lang.String      Address)
                        throws CAENRFIDException
```

C representation:

Not supported

Swift representation:

```
func connect(
                        _ conType: CAENRFIDPort,
                        _ address: String,
                        forceWait: UInt32? = nil
                        ) throws
```

Connect Method (BluetoothSocket)

Description:

Start the android SPP bluetooth communication with the CAEN RFID Reader. This method must be called before any other methods of the Reader object.

Parameters:

Name	Description
BTSock	The BluetoothSocket to read/write data.

Syntax:

Android representation:

```
public void         Connect(
                        BluetoothSocket       BTSock)
                        throws CAENRFIDException
```

C# representation:

Not supported

C representation:

Not supported

Swift representation:

Not supported

Remarks

The BTSock parameter must be obtained trought a *createRfcommSocketToServiceRecord*(UUID uuid) call.

The standard UUID for the Serial Port Profile is 00001101-0000-1000-8000-00805F9B34FB.

Connect Method (BluetoothDevice)

Description:

Start the android SPP bluetooth communication with the CAEN RFID Reader. This method must be called before any other methods of the Reader object.

Parameters:

Name	Description
Context	The application context (default)
BluetoothDevice	The ble device to connect to.

Syntax:

Android representation:

```
public void         Connect(
                        Context       context,
                         BluetoothDevice bluetoothDevice)
                        throws CAENRFIDException
```

C# representation:

Not supported

C representation:

Not supported

Swift representation:

Not supported

Remarks

Before connecting to the BLE device, it should be discovered using the BLEPort.findCAENRFIDBLEDevices (int searchTimeout, CAENRFIDBleScannerCallBacks bleScannerCallbacks) static method. Here an example of how the user can retrieves all BLE devices and add them into an array of BluetoothDevice called“devices”.

```
BLEPort.findCAENRFIDBLEDevices(SEARCH_TIMEOUT, new CAENRFIDBleScannerCallBacks()
{
            @Override
            public void onStartScan() {
                Toast.makeText(
                    getApplicationContext(),
                    " Searching BLE devices", Toast.LENGTH_SHORT)
                .show();
            }

            @Override
            public void onStopScan() {
                Toast.makeText(
                    getApplicationContext(),
                    "Stop searching BLE devices", Toast.LENGTH_SHORT)
                .show();
            }

            @Override
            public void onDeviceFound(BluetoothDevice bluetoothDevice) {
                devices.add(bluetoothDevice);
            }
        });
```

Connect Method (VCPSerialPort)

Description:

Start the android VCP communication with the CAEN RFID Reader. This method must be called before any other methods of the Reader object.

Parameters:

Name	Description
port	The vcp port used to communicate with reader.

Syntax:

Android representation:

```
public void         Connect(
                        VCPSerialPort         port)
                        throws CAENRFIDException
```

C# representation:

Not supported

C representation:

Not supported

Swift representation:

Not supported

Remarks

To find the VCP Port of the CAENRFID USB readers attached to the Android device, use the *VCPSerialPort.findVCPDevice*(Context) class method where the Context parameter is an UsbManager object, and pick the one (usually the first) related to the usb port of the reader.

To use a VCPSerialPort and get an UsbManager objects, the user application should obtain the permission to use the USB System Service of the Android OS.

See the Android USB topics on the developer.android.com site for further details.

ForceAbort Method

Description:

This method tries to stop a pending continuos inventory (see EventInventoryTag Method) that has not been stopped correclty by an InventoryAbort or Disconnect call. Choose the timeout value based on the expected reader load (large value if in presence of a large population of tags, small value if only few tags/seconds must be read).

Parameters:

Name	Description
timeout	The time (in ms) to wait for the end of the continuous inventory data.

Return value:

True if a continuos inventory end has been detected, false otherwise.

Syntax:

C# representation:

```
public bool         ForceAbort(
                                long                    timeout );
```

Java and Android representation:

```
public boolean              ForceAbort(
                                long                    timeout)
                            throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes          CAENRFID_ForceAbort(
                                CAENRFIDHandle          handle,
                            int                     timeout,
                                BOOL                    *endOfStreamMatched);
```

Swift representation:

```
func forceAbort(timeout: UInt64) throws -> Bool
```

Remarks:

If continuos data stream is detected, the ForceAbort waits for its end even if the necessary amount of time exceedes the timeout parameter value.

CAENRFID_Init

Description:

In C language, this function generates an opaque handle to identify a module attached to the PC. See § *Managing connections with the readers* page 8 for more information.

Parameters:

Name	Description
ConType	The communication link to use for the connection.
Address	Depending on ConType parameter: IP address for TCP/IP communications ("xxx.xxx.xxx.xxx"), COM port for RS232 communications ("COMx"), an index for USB communications (not yet supported). To specify a TCP port separate address and port by a semi-colon (ex: "192.168.0.1:2300").
handle	The handle that identifies the device.

Syntax:

C representation:

```
CAENRFIDErrorCodes  CAENRFID_Init(
                            CAENRFIDPort            ConType,
                            char                    *Address,
                            CAENRFIDHandle          *handle,
                            CAENRFIDProtocol        *Protocol);
```

Disconnect Method

Description:

In C# and Java languages, this method closes the connection with the CAEN RFID Reader releasing all the allocated resources. See § *Managing connections with the readers* page 8 for more information.

Syntax:

C# representation:

```
public void         Disconnect()
```

Java and Android representation:

```
public void         Disconnect()
                            throws CAENRFIDException
```

Swift representation:

```
func disconnect() throws
```

CAENRFID_End

Description:

In C language, this function closes the connection with the CAEN RFID Reader releasing all the allocated resources. See § *Managing connections with the readers* page 8 for more information.

Parameters:

Name	Description
handle	The handle that identifies the device.

Syntax:

C representation:

```
CAENRFIDErrorCodes  CAENRFID_End(
                            CAENRFIDHandle        handle);
```

GetBatteryLevel Method

Warning: This method is supported only by readers with battery capacity.

Description:

This method gets the current battery charge.

Return value:

The current charge level expressed in %.

Syntax:

C# representation:

```
public int GetBatteryLevel()
```

Java and Android representation:

```
public int   GetBatteryLevel()
                                  throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes          CAENRFID_GetBatteryLevel (
                                  CAENRFIDHandle        handle,
                                  unsigned int          *Charge);
```

Swift representation:

```
func getBatteryLevel() throws -> UInt
```

GetBitRate Method

Description:

This method gets the current setting of the RF bit rate.

Return value:

The current RF bit rate value.

Syntax:

C# representation:

```
public CAENRFIDBitRate      GetBitRate()
```

Java and Android representation:

```
public CAENRFIDBitRate      GetBitRate()
                                    throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes          CAENRFID_GetBitrate(
                                    CAENRFIDHandle          handle,
                                    CAENRFID_Bitrate        *Bitrate);
```

Swift representation:

```
func getBitRate() throws -> CAENRFIDBitRate
```

GetFirmwareRelease Method

GetFirmwareRelease()

Description:

This method permits to read the release of the firmware loaded into the device.

Return value:

A string representing the firmware release of the device.

Syntax:

C# representation:

```
public string               GetFirmwareRelease()
```

Java and Android representation:

```
public java.lang.String     GetFirmwareRelease()
                                    throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes          CAENRFID_GetFirmwareRelease(
                                    CAENRFIDHandle          handle,
                                    char                    *FWRel);
```

GetFirmwareRelease(Int16)

Description:

This method permits to read the release of the firmware loaded into the device, or internal modules.

Parameters:

Name	Description
Level	The zero-based embedding level index that indicates the module from which gets relative information.

Return value:

A string representing the firmware release of the device.

Syntax:

C# representation:

```
public string               GetFirmwareRelease(short Level)
```

Java and Android representation:

```
public java.lang.String     GetFirmwareRelease(short Level)
                                    throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes          CAENRFID_GetFirmwareReleaseLevel(
                                    CAENRFIDHandle        handle,
                                    short                 level,
                                    char                  *FWRel);
```

Swift representation:

```
func getFWRelease(_ level: UInt16? = nil) throws -> String
```

GetIO Method

Warning: This method is supported only by readers with GPIO.

Description:

This method gets the current digital Input and Output lines status.

Return value:

A bitmask representing the I/O lines status. The format and the meaning of the bits depend on the Reader model. Please refer to the corresponding reader user manual available at www.caenrfid.com.

Syntax:

C# representation:

```
public int          GetIO()
```

Java and Android representation:

```
public int          GetIO()
                            throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_GetIO(
                            CAENRFIDHandle        handle,
                            unsigned int          *IORegister);
```

Swift representation:

```
func getIO() throws -> UInt32
```

GetIODirection Method

Warning: This method is supported only by readers with GPIO.

Description:

This method gets the current I/O direction setting as a bitmask. Each bit represents a I/O line, a value of 0 means that the line is configured as an input, 1 as an output. This setting has a meaning only for those readers with configurable I/O lines.

Return value:

A bitmask representing the I/O setting.

Syntax:

C# representation:

```
public int          GetIODirection()
```

Java and Android representation:

```
public int          GetIODirection()
                            throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_GetIODirection(
                            CAENRFIDHandle        handle,
                            unsigned int          *IODirection);
```

Swift representation:

```
func getIODirection() throws -> UInt32
```

GetFHSSMode Method

Description:

This method gets the current Frequency Hopping status.

Return value:

A zero value if the FHSS is disabled, non-zero value if it is enabled.

Syntax:

C# representation:

```
public short        GetFHSSMode()
```

Java and Android representation:

```
public short        GetFHSSMode()
                            throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_GetFHSSMode(
                            CAENRFIDHandle        handle,
                            unsigned short        *FHSSMode);
```

Swift representation:

```
func getFHSSMode() throws -> Bool
```

GetNetwork Method

Warning: This method is not supported by all readers. For more information, please contact the support at *support@caenrfid.com*.

Description:

Gets the IP v4 network settings of the reader (if supported).

Return value:

A CAENRFIDNewtorkInfo instance containing the IP v4 network settings of the reader.

Syntax:

C# representation:

```
public CAENRFIDNetworkInfo  GetFHSSMode()
```

Java and Android representation:

```
public CAENRFIDNetworkInfo  GetFHSSMode()
                            throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes CAENRFID_GetNetwork(
                            CAENRFIDHandle          handle,
                            char                    *IpAddress,
                            char                    *NetMask,
                            char                    *Gateway);
```

Swift representation:

```
func getFHSSMode() throws -> CAENRFIDNetworkInfo
```

GetPower Method

Description:

This method gets the current setting of the RF power expressed in mW.

Return value:

The current conducted RF power expressed in mW.

Syntax:

C# representation:

```
public int          GetPower()
```

Java and Android representation:

```
public int          GetPower()
                           throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_GetPower(
                           CAENRFIDHandle        handle,
                           unsigned int          *Power);
```

Swift representation:

```
func getPower() throws -> Int
```

GetReaderInfo Method

GetReaderInfo()

Description:

This method permits to read the reader information loaded into the device.

Return value:

The reader information of the device.

Syntax:

C# representation:

```
public CAENRFIDReaderInfo          GetReaderInfo()
```

Java and Android representation:

```
public CAENRFIDReaderInfo          GetReaderInfo()
                                          throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes                 CAENRFID_GetReaderInfo(
                                          CAENRFIDHandle        handle,
                                          char                  *Model,
                                          char                  *SerialNum);
```

GetReaderInfo(Int16)

Description:

This method permits to read the reader information loaded into the device, or internal modules.

Parameters:

Name	Description
Level	The zero-based embedding level index that indicates the module from which gets relative information.

Return value:

The reader information of the device.

Syntax:

C# representation:

```
public CAENRFIDReaderInfo          GetReaderInfo(short Level)
```

Java and Android representation:

```
public CAENRFIDReaderInfo          GetReaderInfo(short Level)
                                          throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes                 CAENRFID_GetReaderInfo(
                                          CAENRFIDHandle        handle,
                                          char                  *Level);
```

Swift representation:

```
func getReaderInfo(_ level:UInt16? = nil) throws -> CAENRFIDReaderInfo
```

GetReadPoints Method

Description:

This method gets the names of the read points (antennas) available in the reader.

Return value:

An array containing the read points (antennas) names available in the reader.

Syntax:

C# representation:

```
public string[]             GetReadPoints()
```

Java and Android representation:

```
public java.lang.String[]   GetReadPoints()
```

C representation:

```
CAENRFIDErrorCodes          CAENRFID_GetReadPoints(
                                   CAENRFIDHandle        handle,
                                   char                  **AntNames [],
                                   int                   *AntNumber);
```

Swift representation:

```
static func getReadPoints() -> [String]
```

GetReadPointPower Method

Warning: This method is not supported by all readers. For more information, please contact the support at *support@caenrfid.com*.

GetReadPointPower(String)

Description:

Gets the conducted power in mW set on antenna identified by readPoint.

Parameters:

Name	Description
readPoint	The name of the antenna from which the power is measured.

Return value:

The conducted power set for the antenna referred in readPoint parameter expressed in mW.

Syntax:

C# representation:

```
public int          GetReaderPointPower(String ReadPoint)
```

Java and Android representation:

```
public int          GetReaderPointPower(String ReadPoint)
                                        throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes          CAENRFID_GetReaderPointPower(
                                        CAENRFIDHandle      handle,
                                        char                *readPoint,
                                        int                 *power);
```

Swift representation:

```
func                        getReadPointPower(_ readPoint:String) throws -> Int
```

GetReadPointStatus Method

Description:

This method gets the CAENRFIDReadPointStatus object representing the status of a read point (antenna).

Parameters:

Name	Description
ReadPoint	The name of the read point to check.

Return value:

The CAENRFIDReadPointStatus object representing the current status of the read point.

Syntax:

C# representation:

```
public CAENRFIDReadPointStatus      GetReadPointStatus(
                                        string                      ReadPoint)
```

Java and Android representation:

```
public CAENRFIDReadPointStatus      GetReadPointStatus(
                                        java.lang.String            ReadPoint)
                                        throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes                  CAENRFID_GetReadPointStatus(
                                        CAENRFIDHandle              handle,
                                        char                        *ReadPoint,
                                        CAENRFIDReadPointStatus     *Status);
```

Swift representation:

```
func getReadPointStatus(_ readPoint: String = "Ant0") throws -> CAENRFIDReadPointStatus
```

GetRFChannel Method

Description:

This method gets the index of the RF channel currently in use. The index value meaning changes for different country regulations.

Return value:

The RF channel index.

Syntax:

C# representation:

```
public short        GetRFChannel()
```

Java and Android representation:

```
public short        GetRFChannel()
                            throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_GetRFChannel(
                            CAENRFIDHandle      handle,
                            unsigned short      *RFChannel);
```

Shift representation:

Not supported

Remarks

This method is only used for testing applications.

GetRFRegulation Method

Description:

This method gets the current RF regulation setting value.

Return value:

The RF regulation value.

Syntax:

C# representation:

```
public CAENRFIDRFRegulations        GetRFRegulation()
```

Java and Android representation:

```
public CAENRFIDRFRegulations        GetRFRegulation()
                                    throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes                  CAENRFID_GetRFRegulation(
                                            CAENRFIDHandle              handle,
                                            CAENRFIDRFRegulations       *RFRegulation);
```

Swift representation:

```
func getRFRegulation() throws -> CAENRFIDRFRegulations
```

GetSource Method

Description:

This method gets a CAENRFIDLogicalSource object given its name.

Parameters:

Name	Description
Source	The name of the logical source.

Return value:

The CAENRFIDLogicalSource object corresponding to the requested name.

Syntax:

C# representation:

```
public CAENRFIDLogicalSource        GetSource(
                                            string              Source)
```

Java and Android representation:

```
public CAENRFIDLogicalSource        GetSource(
                                            java.lang.String    Source)
                                            throws CAENRFIDException
```

C representation:

Not supported. This function does not exist in C language, see § *Overview on SDK* page 7 for more information.

Swift representation:

```
func getSource(_ source: String) throws -> CAENRFIDLogicalSources?
```

GetSourceNames Method

Description:

This method gets the names of the logical sources available in the reader.

Return value:

An array containing the logical source names available in the reader.

Syntax:

C# representation:

```
public static string[]              GetSourceNames()
```

Java and Android representation:

```
public static java.lang.String[]    GetSourceNames()
```

C representation:

```
CAENRFIDErrorCodes                  CAENRFID_GetSourceNames(
                                            CAENRFIDHandle          handle,
                                            char                    **SrcNames[],
                                            int                     *SrcNumber);
```

Swift representation:

```
static func getSourceNames() -> [String]
```

GetSources Method

Description:

This method gets the CAENRFIDLogicalSource objects available on the reader.

Return value:

An array of the logical source objects available in the Reader.

Syntax:

C# representation:

```
public CAENRFIDLogicalSource[]      GetSources()
```

Java and Android representation:

```
public CAENRFIDLogicalSource[]      GetSources()
```

C representation:

Not supported. This function does not exist in C language, see § *Overview on SDK* page 7 for more information.

Swift representation:

```
static func getSources() -> [CAENRFIDLogicalSources?]?
```

InventoryAbort Method

Description:

This method stops the EventInventoryTag execution.

Syntax:

C# representation:

```
public void                 InventoryAbort()
```

Java and Android representation:

```
public void                 InventoryAbort()
                            throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes          CAENRFID_InventoryAbort(
                                    CAENRFIDHandle          handle);
```

Swift representation:

```
func inventoryAbort() throws
```

MatchReadPointImpedance Method

> **Warning**: This method is not supported by all readers. For more information, please contact the support at *support@caenrfid.com*.

MatchReadPointImpedance (String)

Description:

MatchReadPointImpedance matches the antenna impedance passed in ReadPoint.

Parameters:

Name	Description
ReadPoint	The antenna to be matched

Return value:

A real number greater than one, that represents the return status of the matching operation.

Syntax:

C# representation:

```
public float        MatchReadPointImpedance(
                        string              ReadPoint)
```

Java and Android representation:

```
public float        MatchReadPointImpedance(
                        String              ReadPoint)
                        throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_ MatchReadPointImpedance (
                        CAENRFIDHandle      handle,
                        char                *ReadPoint,
                        float               *Value);
```

Swift representation:

Not supported

MatchReadPointImpedance (String, CAENRFIDMatchingParams, Int16)

Description:

MatchReadPointImpedance matches the antenna impedance passed in ReadPoint.

Parameters:

Name	Description
ReadPoint	The antenna to be matched
MatchParam	A CAENRFIDMatchingParams parameters for matching operation
MatchParamValue	The value of the MatchParam

Return value:

A real number greater than one, that represents the return status of the matching operation.

Syntax:

C# representation:

```
public float        MatchReadPointImpedance (
                        string                  ReadPoint,
                        CAENRFIDMatchingParams  MatchParam,
                        short                   MatchParamValue)
```

Java and Android representation:

```
public float        MatchReadPointImpedance(
                        String                      ReadPoint,
                        CAENRFIDMatchingParams      MatchParam,
                        short                       MatchParamValue)
                        throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_ MatchReadPointImpedance (
                        CAENRFIDHandle              handle,
                        char                        *ReadPoint,
                        CAENRFIDMatchingParams      MatchParam,
                        int                         MatchParamValue,
                        float                       *Value);
```

Swift representation:

```
func matchReadPointImpedance(_ readPoint: String,
                        matchParam: CAENRFIDMatchingParams,
                        matchParamValue: UInt32)
                        throws -> Float
```

SetBitRate Method

Description:

This method sets the RF bit rate to use.

Parameters:

Name	Description
BitRate	The RF bit rate value to be set.

Syntax:

C# representation:

```
public void         SetBitRate(
                        CAENRFIDBitRate     BitRate)
```

Java and Android representation:

```
public void         SetBitRate(
                        CAENRFIDBitRate     BitRate)
                        throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_SetBitRate(
                        CAENRFIDHandle      handle,
                        CAENRFID_Bitrate    BitRate);
```

Swift representation:

```
func setBitRate(bitRate: CAENRFIDBitRate)
                    throws
```

SetDateTime Method

Warning: This method is not supported by all readers. For more information, please contact the support at *support@caenrfid.com*.

Description:

This method sets the Date/Time of the reader.

Parameters:

Name	Description
DateTime	The Date/Time to be set on the reader as a string in the format: "yyyy-mm-dd hh:mm:ss".

Syntax:

C# representation:

```
public void         SetDateTime(
                            string                  DateTime)
```

Java and Android representation:

```
public void         SetDateTime(
                            java.lang.String        DateTime)
                            throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_SetDateTime(
                            CAENRFIDHandle          handle,
                            char                    *DateTime);
```

Swift representation:

```
func setDateTime(dateTime: String)
                    throws
```

SetIO Method

Warning: This method is supported only by readers with GPIO.

Description:

This method sets the Output lines value.

Parameters:

Name	Description
IOValue	A bitmask representing the I/O lines value. The format and the meaning of the bits depend on the reader model. Please refer to the corresponding user manual available on www.caenrfid.com

Syntax:

C# representation:

```
public void         SetIO(
                            int                     IOValue)
```

Java and Android representation:

```
public void         SetIO(
                            int                     IOValue)
                            throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_SetIO(
                            CAENRFIDHandle          handle,
                            unsigned int            IOValue);
```

Swift representation:

```
func                setIO(ioValue: UInt32)
          throws
```

SetIODIRECTION Method

Warning: This method is supported only by readers with GPIO.

Description:

This method sets the current I/O direction setting as a bitmask. Each bit represents a I/O line, a value of 0 means that the line is configured as an input, 1 as an output. This setting has a meaning only for those readers with configurable I/O lines.

Parameters:

Name	Description
IODirection	The IODirection value to set.

Syntax:

C# representation:

```
public void         SetIODIRECTION(
                            int                     IODirection)
```

Java and Android representation:

```
public void         SetIODIRECTION(
                            int                     IODirection)
                            throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_SetIODirection(
                            CAENRFIDHandle          handle,
                            unsigned int            IODirection);
```

Swift representation:

```
func setIODirection(ioDirection: UInt32)
           throws
```

SetPower Method

Description:

This method sets the conducted RF power of the Reader.

Parameters:

Name	Description
power	The conducted RF power value expressed in mW.

Syntax:

C# representation:

```
public void         SetPower(
                            int                     power)
```

Java and Android representation:

```
public void         SetPower(
                            int                     power)
                            throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_SetPower(
                            CAENRFIDHandle          handle,
                            unsigned int            Power);
```

Swift representation:

```
func setPower(_ power: Int) throws
```

SetReadPointPower

Warning: This method is not supported by all readers. For more information, please contact the support at *support@caenrfid.com*.

Description:

Sets the conducted power in mW on antenna identified by readPoint.

Parameters:

Name	Description
readPoint	The name of the antenna from which the power is measured.
power	The conducted power expressed in mW

Syntax:

C# representation:

```
public int          SetReaderPointPower(String ReadPoint, int Power)
```

Java and Android representation:

```
public int          SetReaderPointPower(String ReadPoint, int Power)
                                        throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_SetReaderPointPower(
                                        CAENRFIDHandle      handle,
                                        char                *readPoint,
                                        int                 power);
```

Swift representation:

```
func                setReadPointPower( readPoint:String,  power:Int) throws
```

SetRS232 Method

Warning: This method is supported only by those readers equipped with UART or RS232 interface

Description:

This method permits to change the serial port settings. Valid settings values depend on the reader model.

Parameters:

Name	Description
baud	The baud rate value to set.
datab	The number of data bits to set.
stopb	The number of stop bits to set.
parity	The parity value to set.
flowc	The flow control value to set.

Syntax:

C# representation:

```
public void         SetRS232(
                            int                         baud,
                            int                         datab,
                            int                         stopb,
                            CAENRFIDRS232Constants      parity,
                            CAENRFIDRS232Constants      flowc)
```

Java and Android representation:

```
public void         SetRS232(
                            int                         baud,
                            int                         datab,
                            int                         stopb,
                            CAENRFIDRS232Constants      parity,
                            CAENRFIDRS232Constants      flowc)
                            throws CAENRFIDException
```

C representation:

```
CAENRFIDErrorCodes  CAENRFID_SetRS232(
                            CAENRFIDHandle              handle,
                            unsigned long               baud,
                            unsigned long               datab,
                            unsigned long               stopb,
                            CAENRFID_RS232_Parity       parity,
                            CAENRFID_RS232_FlowControl  flowc);
```

Swift representation:

```
func setRS232(
                            _ baud: UInt = 115200,
                            _ dataB: UInt = 8,
                            _ stop: UInt = 1,
                            _ parity: CAENRFIDRS232Constants = .CAENRFID_RS232_Parity_None,
                            _ flowc: CAENRFIDRS232Constants =
.CAENRFID_RS232_FlowControl_None) throws
```

CAENRFIDReaderInfo Class

The CAENRFIDReaderInfo class is used to create reader info objects. Reader info objects represent the information about the reader device (model and serial number).

GetModel Method

Description:

This method gets the reader model.

Return value:

The reader model.

Syntax:

C# representation:

```
public string                       GetModel()
```

Java and Android representation:

```
public java.lang.String             GetModel()
```

C representation:

Not supported. This method does not exist in C language. It is possible to use the *GetReaderInfo Method* page 94 instead. In fact *GetReaderInfo Method* (in the C language) returns the reader model and the serial number.

Swift representation:

```
var model: String {get }
```

GetSerialNumber Method

Description:

This method gets the reader serial number.

Return value:

The reader serial number.

Syntax:

C# representation:

```
public string                       GetSerialNumber()
```

Java and Android representation:

```
public java.lang.String             GetSerialNumber()
```

C representation:

Not supported. This method does not exist in C language. It is possible to use the *GetReaderInfo Method* page 94 instead. In fact *GetReaderInfo Method* (in the C language) returns the reader model and the serial number.

Swift representation:

```
var serialNumber: String { get }
```

CAENRFIDTag Class

The CAENRFIDTag class is used to define objects representing the tags. These objects are used as return values for the inventory methods and as arguments for many tag access methods.

In both Java and C# language this class is composed by methods while in C language the following struct is present (for more information see § *Overview on SDK* page 2):

C representation:

```
typedef struct {
                    byte                ID[MAX_ID_LENGTH];
                    short               Length;
                    char                LogicalSource[MAX_LOGICAL_SOURCE_NAME];
                    char                ReadPoint[MAX_READPOINT_NAME];
                    CAENRFIDProtocol    Type;
                    short               RSSI;
                    byte                TID[MAX_TID_SIZE];
                    short               TIDLen;
                    byte                XPC[XPC_LENGTH];
                    byte                PC[PC_LENGTH];
            } CAENRFIDTag;
```

GetId Method

Description:

This method returns the tag ID (the EPC code in Gen2 tags).

Return value:

An array of bytes representing the tag ID (the EPC code in EPC Class 1 Gen 2 tags).

Syntax:

C# representation:

```
public byte[]               GetId()
```

Java and Android representation:

```
public byte[]               GetId()
```

C representation:

Not supported

Swift representation:

```
func getId() -> [UInt8]
```

GetLength Method

Description:

This method returns the tag ID length.

Return value:

The tag length.

Syntax:

C# representation:

```
public short                GetLength()
```

Java and Android representation:

```
public short                GetLength()
```

C representation:

Not supported

Swift representation:

```
func getLength() -> UInt16
```

GetPC Method

Description:

This method returns the Protocol Control(PC) word code of the tag.

Return value:

The tag Protocol Control code.

Syntax:

C# representation:

```
public byte[]               GetPC()
```

Java and Android representation:

```
public byte[]               GetPC()
```

C representation:

Not supported

Swift representation:

```
func getPC() -> [UInt8]
```

GetReadPoint Method

Description:

This method returns the read point that has detected the tag.

Return value:

The name of the read point that has detected the Tag

Syntax:

C# representation:

```
public string                       GetReadPoint()
```

Java and Android representation:

```
public java.lang.String             GetReadPoint()
                                    throws CAENRFIDException
```

C representation:

Not supported

Swift representation:

```
func getReadPoint() -> String
```

GetRSSI Method

Description:

This method returns the RSSI value measured for the tag.

Return value:

The tag RSSI.

Syntax:

C# representation:

```
public short                        GetRSSI()
```

Java and Android representation:

```
public short                        GetRSSI()
```

C representation:

Not supported

Swift representation:

```
func getRSSI() -> Int16
```

GetSource Method

Description:

This method returns the name of the logical source that has detected the tag.

Return value:

The name of the logical source that has detected the tag.

Syntax:

C# representation:

```
public CAENRFIDLogicalSource       GetSource()
```

Java and Android representation:

```
public CAENRFIDLogicalSource       GetSource()
```

C representation:

Not supported

Swift representation:

```
func getSource() -> CAENRFIDLogicalSource
```

GetTID Method

Description:

This method returns the tag TID (valid only for EPC Class 1 Gen 2 tags).

Return value:

An array of bytes representing the tag TID.

Syntax:

C# representation:

```
public byte[]              GetTID()
```

Java and Android representation:

```
public byte[]              GetTID()
```

C representation:

Not supported

Swift representation:

```
func getTID() -> [UInt8]?
```

GetTimeStamp Method

Description:

This method gets the Tag TimeStamp.

Return value:

The Tags's Unix TimeStamp.

Syntax:

C# representation:

```
public DateTime                    GetTimeStamp()
```

Java and Android representation:

```
public java.util.Date              GetTimeStamp()
```

C representation:

Not supported

Swift representation:

```
func getTimeStamp() -> Date
```

GetType Method

Description:

This method returns the air protocol of the tag.

Return value:

The air protocol of the tag.

Syntax:

C# representation:

```
public new CAENRFIDProtocol        GetType()
```

Java and Android representation:

```
public CAENRFIDProtocol            GetType()
```

C representation:

Not supported

Swift representation:

```
func getType() -> CAENRFIDProtocol
```

GetXPC Method

Description:

This method returns the tag XPC words.

Return value:

The tag XPC words.

Syntax:

C# representation:

```
public byte[]              GetXPC()
```

Java and Android representation:

```
public byte[]              GetXPC()
```

C representation:

Not supported

Swift representation:

```
func getXPC() -> [UInt8]?
```

getFrequency Method

Description:

Get the backscattering frequency.

Return value:

An integer value representing the frequency expressed in KHz.

Syntax:

C# representation:

```
public long         getFrequency()
```

Java and Android representation:

```
public long         getFrequency()
```

C representation:

Not supported

Swift representation:

```
func getFrequency() -> UInt32
```

getPhaseBegin Method

Description:

Gets the backscattering phase at the begin of communication from reader to tag.

Return value:

A decimal value expressed in Degree.

Syntax:

C# representation:

```
public float        getPhaseBegin()
```

Java and Android representation:

```
public float        getPhaseBegin()
```

C representation:

Not supported

Swift representation:

```
func getPhaseBegin() -> Float
```

getPhaseEnd Method

Description:

Gets the backscattering phase at the end of communication from reader to tag.

Return value:

A decimal value expressed in Degree.

Syntax:

C# representation:

```
public float        getPhaseEnd()
```

Java and Android representation:

```
public float        getPhaseEnd()
```

C representation:

Not supported

Swift representation:

```
func getPhaseEnd() -> Float
```

7 ENUMERATIONS DESCRIPTION

CAENRFIDBitRate Enumeration

The CAENRFIDBitRate Enumeration gives a list of the supported radiofrequency profiles.

Syntax:

C# representation:

```
public enum                CAENRFIDBitRate
```

Java and Android representation:

```
public final class         CAENRFIDBitRate
```

C representation:

```
typedef enum               CAENRFID_Bitrate;
```

Swift representation:

```
public enum                CAENRFIDBitRate: UInt16
```

In the following table, the CAENRFIDBitRate Enumeration members are listed:

Member	Description
DSB_ASK_FM0_TX10RX40	DSB-ASK transmission modulation, FM0 return link encoding, 10 Kbit in transmission, 40 Kbit in reception.
DSB_ASK_FM0_TX40RX40	DSB-ASK transmission modulation, FM0 return link encoding, 40 Kbit in transmission, 40 Kbit in reception.
DSB_ASK_FM0_TX40RX160	DSB-ASK transmission modulation, FM0 return link encoding, 40 Kbit in transmission, 160 Kbit in reception.
DSB_ASK_FM0_TX160RX400	DSB-ASK transmission modulation, FM0 return link encoding, 160 Kbit in transmission, 400 Kbit in reception.
DSB_ASK_M2_TX40RX160	DSB-ASK transmission modulation, Miller (M=2) return link encoding, 40 Kbit in transmission, 160 Kbit in reception.
DSB_ASK_M4_TX40RX256	DSB-ASK transmission modulation, Miller (M=4) return link encoding, 40 Kbit in transmission, 256 Kbit in reception.
PR_ASK_FM0_TX40RX640	PR-ASK transmission modulation, FM0 return link encoding, 40 Kbit in transmission, 640 Kbit in reception.
PR_ASK_M2_TX40RX250	PR-ASK transmission modulation, Miller (M=2) return link encoding, 40 Kbit in transmission, 250 Kbit in reception.
PR_ASK_M4_TX40RX250	PR-ASK transmission modulation, Miller (M=4) return link encoding, 40 Kbit in transmission, 250 Kbit in reception.
PR_ASK_M4_TX40RX256	PR-ASK transmission modulation, Miller (M=4) return link encoding, 40 Kbit in transmission, 256 Kbit in reception.
PR_ASK_M4_TX40RX300	PR-ASK transmission modulation, Miller (M=4) return link encoding, 40 Kbit in transmission, 300 Kbit in reception.
PR_ASK_M4_TX40RX320	DSB-ASK transmission modulation, Miller (M=4) return link encoding, 40 Kbit in transmission, 320 Kbit in reception.
PR_ASK_M4_TX80RX320	PR-ASK transmission modulation, Miller (M=4) return link encoding, 80 Kbit in transmission, 320 Kbit in reception.
DSB_ASK_M8_TX40RX256	DSB-ASK transmission modulation, Miller (M=8) return link encoding, 40 Kbit in transmission, 256 Kbit in reception.
DSB_ASK_FM0_TX160RX640	DSB-ASK transmission modulation, FM0 return link encoding, 160 Kbit in transmission, 640 Kbit in reception.
DSB_ASK_M2_TX160RX640	DSB-ASK transmission modulation, Miller (M=2) return link encoding, 160 Kbit in transmission, 640 Kbit in reception.

PR_ASK_FM0_TX133RX640	PR-ASK transmission modulation, FM0 return link encoding, 133 Kbit in transmission, 640 Kbit in reception.
PR_ASK_FM0_TX66RX426	PR-ASK transmission modulation, FM0 return link encoding, 66 Kbit in transmission, 426 Kbit in reception.
PR_ASK_M2_TX133rx640	PR-ASK transmission modulation, Miller (M=2) return link encoding, 133 Kbit in transmission, 640 Kbit in reception.
PR_ASK_M2_TX50RX320	PR-ASK transmission modulation, Miller (M=2) return link encoding, 50 Kbit in transmission, 320 Kbit in reception.
PR_ASK_M2_TX66RX320	PR-ASK transmission modulation, Miller (M=2) return link encoding, 66 Kbit in transmission, 320 Kbit in reception.
PR_ASK_M4TX133RX640	PR-ASK transmission modulation, Miller (M=4) return link encoding, 133 Kbit in transmission, 640 Kbit in reception.
PR_ASK_M4_TX50RX250	PR-ASK transmission modulation, Miller (M=4) return link encoding, 50 Kbit in transmission, 250 Kbit in reception.
PR_ASK_M4_TX50RX320	PR-ASK transmission modulation, Miller (M=4) return link encoding, 50 Kbit in transmission, 320 Kbit in reception.
PR_ASK_M8_TX50RX160	PR-ASK transmission modulation, Miller (M=8) return link encoding, 50 Kbit in transmission, 160 Kbit in reception.

CAENRFIDLogicalSourceConstants Enumeration

The CAENRFIDLogicalSourceConstants Enumeration gives a list of constants used for the configuration of the logical sources. Detailed explanation of the settings can be found in the EPC Class 1 Gen 2 and ISO 18000-6B specification documents.

Syntax:

C# representation:

```
public enum                CAENRFIDLogicalSourceConstants
```

Java and Android representation:

```
public final class         CAENRFIDLogicalSourceConstants
```

C representation:

```
typedef enum               CAENRFIDLogicalSourceConstants;
```

Swift representation:

```
public enum                CAENRFIDLogicalSourceConstants
```

In the following table, the CAENRFIDLogicalSourceConstants Enumeration members are listed:

Member	Description
EPC_C1G2_SESSION_S0	Session 0 is selected for the anticollision algorithm execution on the logical source (valid only for the EPC Class1 Gen2 air protocol).
EPC_C1G2_SESSION_S1	Session 1 is selected for the anticollision algorithm execution on the logical source (valid only for the EPC Class1 Gen2 air protocol).
EPC_C1G2_SESSION_S2	Session 2 is selected for the anticollision algorithm execution on the logical source (valid only for the EPC Class1 Gen2 air protocol).
EPC_C1G2_SESSION_S3	Session 3 is selected for the anticollision algorithm execution on the logical source (valid only for the EPC Class1 Gen2 air protocol).
EPC_C1G2_TARGET_A	Target A is selected for the anticollision algorithm execution on the logical source (valid only for the EPC Class1 Gen2 air protocol).
EPC_C1G2_TARGET_B	Target B is selected for the anticollision algorithm execution on the logical source (valid only for the EPC Class1 Gen2 air protocol).
EPC_C1G2_SELECTED_YES	Only the tags with the SL flag set to true are considered in the anticollision algorithm execution on the logical source (valid only for the EPC Class1 Gen2 air protocol).
EPC_C1G2_SELECTED_NO	Only the tags with the SL flag set to false are considered in the anticollision algorithm execution on the logical source (valid only for the EPC Class1 Gen2 air protocol).
EPC_C1G2_ALL_SELECTED	All the tags are considered in the anticollision algorithm execution on the logical source (valid only for the EPC Class1 Gen2 air protocol).

CAENRFIDLogicalSource.InventoryFlag Enumeration

The CAENRFIDLogicalSource.InventoryFlag Enumeration gives a list of constants used for the configuration of the inventory function that comes with Flag parameter.

Syntax:

C# representation:

```
public enum                CAENRFIDLogicalSource.InventoryFlag
```

Java and Android representation:

```
public final class         CAENRFIDLogicalSource.InventoryFlag
```

C representation:

```
typedef enum               CAENRFIDLogicalSource.InventoryFlag;
```

Swift representation:

```
public enum                InventoryFlag:UInt16
```

In the following table, the CAENRFIDLogicalSource.InventoryFlag Enumeration members are listed:

Member	Description
RSSI	When enabled, the RSSI value representing the backscattered RF field strenght is returned by the reader for each tag read. Some reader cannot have this feature.
FRAMED	Tags found in an inventory cycle are not buffered in reader and sent all together, but sent one by one as soon as a tag is detected. It Is used in conjunction with the continuous flag.
CONTINUOUS	Enables the continuous mode acquisition. Logical source must have ReadCycle parameter set to 0.
COMPACT	Instruct the reader to not return any other information than the ID. Other values are fake and filled by the library.
TID_READING	Instruct the reader to return the TID memory. On some reader it must be used in conjunction with SetTIDLength to work more efficiently.
EVENT_TRIGGER	Work only in combination with continuous mode. In reader provided with identification button, it instructs the reader to do an inventory cycle only when the button is pressed.
XPC	It instructs the reader to return XPC. If no XPC is present on the tag, the XPC field of a tag is filled up with zero values.
PC	Instruct the reader to return the PC of the EPC bank for each inventoried tag.
PHASE	When enabled the reader returns in each identified tag the relative phase begin and end of the backscattering signal during its identification.
FREQUENCY	When enabled the reader returns in each identified tag the relative backscattering frequency during its identification.

CAENRFIDPort Enumeration

The CAENRFIDPort Enumeration gives a list of the communication ports supported by the CAEN RFID readers.

Syntax:

C# representation:

```
public enum                CAENRFIDPort
```

Java and Android representation:

```
public final class         CAENRFIDPort
```

C representation:

```
typedef enum               CAENRFIDPort;
```

Swift representation:

```
public enum CAENRFIDPort
```

Remarks:

In order to align the three libraries, the members name in C language have changed, now reporting the CAENRFID_ suffix, but the value of the members is the same of the previous library version.

In the following table, the CAENRFIDPort Enumeration members are listed:

Member	Description
CAENRFID_RS232	Serial port communication link (used either for USB-Serial and Bluetooth emulated COM port on Desktop OS only).
CAENRFID_TCP	TCP/IP network communication link.
CAENRFID_USB	USB communication link (Android only)
CAENRFID_BLUETOOTH	Serial port emulated via classic RFComm (Android) or ExternalAccessory(iOS)
CAENRFID_BLE	BLE communication link (Android and iOS only)

CAENRFIDProtocol Enumeration

The CAENRFIDProtocol Enumeration gives a list of the air protocol supported by the CAEN RFID readers.

Syntax:

C# representation:

```
public enum                 CAENRFIDProtocol
```

Java and Android representation:

```
public final class          CAENRFIDProtocol
```

C representation:

```
typedef enum                CAENRFIDProtocol;
```

Swift representation:

```
public enum                 CAENRFIDProtocol:UInt16
```

Remarks:

In order to align the three libraries, the members name in C language have changed, now reporting the CAENRFID_ suffix, but the value of the members is the same of the previous library version.

In the following table, the CAENRFIDProtocol Enumeration members are listed:

Member	Description
CAENRFID_ISO18000_6b	ISO18000-6B air protocol.
CAENRFID_EPC119	EPC 1.19 air protocol.
CAENRFID_EPC_C1G1	EPCGlobal Class1 Gen1 air protocol.
CAENRFID_ISO18000_6a	ISO18000-6A air protocol.
CAENRFID_EPC_C1G2	EPCGlobal Class1 Gen2 (aka ISO18000-6C) air protocol. (DEFAULT)
CAENRFID_MULTIPROTOCOL	This value permits to use all the supported air protocol at the same time. Suggested setting only for demo purposes.

CAENRFIDReadPointStatus Enumeration

The CAENRFIDReadPointStatus gives a list of the possible ReadPoint status values.

Syntax:

C# representation:

```
public enum                 CAENRFIDReadPointStatus
```

Java and Android representation:

```
public final class          CAENRFIDReadPointStatus
```

C representation:

```
typedef enum                CAENRFIDReadPointStatus;
```

Swift representation:

```
public enum                 CAENRFIDReadPointStatus:UInt16
```

Remarks:

In order to align the three libraries, the members name in C language have changed, now reporting the STATUS_ suffix, but the value of the members is the same of the previous library version.

In the following table, the CAENRFIDReadPointStatus Enumeration members are listed:

Member	Description
STATUS_BAD	Bad antenna connection.
STATUS_GOOD	Good antenna connection.
STATUS_POOR	Poor antenna connection.

CAENRFIDRFRegulations Enumeration

The CAENRFIDRFRegulations gives a list of country radiofrequency regulations.

Syntax:

C# representation:

```
public enum                CAENRFIDRFRegulations
```

Java and Android representation:

```
public final class         CAENRFIDRFRegulations
```

C representation:

```
typedef enum               CAENRFIDRFRegulations;
```

Swift representation:

```
public enum                CAENRFIDRFRegulations:UInt16
```

Remarks:

In order to align the three libraries, the regulations, previously declared as #define, are now members of an enumeration, but the value of the members is the same of the previous library version.

In the following table, the CAENRFIDRFRegulations Enumeration members are listed:

Member	Description
ETSI_302208	ETSI_302208 radiofrequency regulation.
ETSI_300220	ETSI_300220 radiofrequency regulation.
FCC_US	FCC_US radiofrequency regulation.
MALAYSIA	MALAYSIA radiofrequency regulation.
JAPAN	JAPAN radiofrequency regulation.
KOREA	KOREA radiofrequency regulation.
AUSTRALIA	AUSTRALIA radiofrequency regulation.
CHINA	CHINA radiofrequency regulation.
TAIWAN	TAIWAN radiofrequency regulation.
SINGAPORE	SINGAPORE radiofrequency regulation.
BRAZIL	BRAZIL radiofrequency regulation.
JAPAN_STD_T106 11	JAPAN radiofrequency regulation (ARIB STD-T106 Premises radio station (1W) - LBT free)
JAPAN_STD_T107 12	JAPAN radiofrequency regulation (ARIB STD-T107 Specified low power radio station (250mW) - with LBT)
JAPAN_STD_106_L	JAPAN radiofrequency regulation (ARIB STD-T106 Specified low power radio station (250mW) - with LBT)
PERU	PERU radiofrequency regulation.
SOUTH_AFRICA	SOUTH AFRICA radiofrequency regulation.
CHILE	CHILE radiofrequency regulation
HONG KONG	HONG KONG radiofrequency regulation
INDONESIA	INDONESIA radiofrequency regulation
NEW_ZELAND	NEW_ZELAND radiofrequency regulation
RUSSIA	RUSSIA radiofrequency regulation
ISRAEL	ISRAEL radiofrequency regulation
ETSI_302208_UB	UB = Upper Band.

CAENRFIDRS232Constants Enumeration

The CAENRFIDRS232Constants gives a list of settings for the serial port configuration.

Syntax:

C# representation:

```
public enum                 CAENRFIDRS232Constants
```

Java and Android representation:

```
public final class          CAENRFIDRS232Constants
```

C representation:

```
typedef enum                CAENRFID_RS232_Parity;
typedef enum                CAENRFID_RS232_FlowControl;
```

Swift representation:

```
public enum                 CAENRFIDRS232Constants
```

In the following table, the CAENRFIDRS232Constants Enumeration members are listed:

Member	Description
CAENRS232_Parity_None	No parity bit is sent at all.
CAENRS232_Parity_Odd	Odd parity.
CAENRS232_Parity_Even	Even parity.
CAENRFID_RS232_FlowControl_XonXoff	Software flow control.
CAENRFID_RS232_FlowControl_Hardware	Hardware flow control.
CAENRFID_RS232_FlowControl_None	No flow control.

CAENRFIDTag.MemBanks Enumeration

The CAENRFIDTag.MemBanks enumerates the bank name of a generic ISO18000-6C tag.

Syntax:

C# representation:

```
public enum         MemBanks {
                            RESERVED = 0,
                            EPC = 1,
                            TID = 2,
                            USER = 3
                            }
```

Java and Android representation:

```
public enum         MemBanks {
                            RESERVED(0), EPC(1), TID(2), USER(3);
                            private int code;
                            private MemBanks(int c) {
                            code = c;
                            }
                            public int getBankNum() {
                            return code;
                            }
                            }
```

C representation:

```
typedef enum        {
                            RESERVED    = 0,
                            EPC         = 1,
                            TID         = 2,
                            USER        = 3
                            } CAENRFIDMemBanks;
```

Swift representation:

```
enum        MemBanks {
                            case reserved
                            case epc
                            case tid
                            case user
                            }
```

In the following table, the CAENRFIDTag.MemBanks Enumeration members are listed:

Member	Description
RESERVED	Indicates the reserved bank
EPC	Indicates the EPC bank
TID	Indicates the TID bank
USER	Indicates the USER bank

8 OBSOLETE METHODS

Below it is available a list of the obsolete methods and functions for the different program languages (C#, C, Java and Android).

Warning: It is recommended not to use these methods since they will not be available in new readers firmware release.

- BlockWriteTagData
- CustomCommand_EPC_C1G2
- EventInventoryTag(byte[], Int16, Int16, Int16, InventorySubCommand)
- GetDESB_ISO180006B
- GetProtocol
- getSubCommand
- getSubCommandData
- getSubCommandResultCode
- GroupSelUnsel
- InventoryTag Method (Byte[], Int16, Int16, InventorySubCommand)
- InventoryTag Method (Byte[], Int16, Int16, Int16, InventorySubCommand)
- InventoryTag Method (Int16, Byte[], Int16, Int16, Int16, InventorySubCommand)
- InventoryTag Method (Int16, Byte[], Int16, Int16, InventorySubCommand)
- InventoryTag Method (InventorySubCommand)
- KillTag_EPC_C1G1
- LockTag_ISO180006B
- NXP_ChangeConfig
- NXP_ChangeEAS
- NXP_EAS_Alarm
- NXP_ReadProtect
- NXP_ResetReadProtect
- PrintScreen
- ProgramID_EPC_C1G1
- ProgramID_EPC119
- Query_EPC_C1G2
- QueryAck_EPC_C1G2
- ReadTagData
- ResetSession_EPC_C1G2
- RFControl
- SetDESB_ISO180006B
- SetNetwork
- SetProtocol
- SetRFChannel
- SL900A_EndLog
- SL900A_GetLogState
- SL900A_GetMeasurementSetup
- SL900A_GetSensorValue
- SL900A_Initialize
- SL900A_SetLogLimits
- SL900A_SetLogMode
- SL900A_StartLog
- WriteTagData