

RFID Reader API for .NET Developers

API Reference Manual

2009/7

Version 1.7.1

Model : NL-RD600, NL-RD400

Revision History

Version 1.7.1

- 작성일 : 2009 년 7 월 17 일
- 수정내용
- ★ 초안

Version #.#.#

- 작성일 :
- 수정내용
- ★

Contents

1. INTRODUCTION

- 1.1 Purpose
- 1.2 Overview
- 1.3 Architectures

2. QUICK START

- 2.1 리더 콘트롤 참조 추가
- 2.2 리더 콘트롤 생성 및 초기화
- 2.3 연결 처리
- 2.4 연결 해지 및 종료
- 2.5 이벤트 핸들러 작성
- 2.6 태그 ID 읽기 – Inventory
- 2.7 동작 중지 – Stop operations

3. READER API

- 3.1 CONSTRUCTORS
- 3.2 PROPERTIES
- 3.3 EVENTS
- 3.4 METHODS

1. INTRODUCTION

1.1 Purpose

이 문서는 (주)네스랩 “**RFID 동글 또는 모듈형 리더**” 를 적용하고자 하는 응용 프로그램 개발자 혹은 시스템 통합자들에게 필요한 리더 API 를 기술하는 데 그 목적이 있다.

패킷 규격 및 명령어 상세 내용은 별도로 작성된 “Reader API Specifications” 문서를 참조 합니다.

The intended audience for this document includes the following:

- RFID middleware or application software developers who will be creating software for configuring, controlling, and accessing the RFID reader

1.2 Overview

이 문서에서 기술되는 **리더 콘트롤 라이브러리**는 Visual Studio .NET 개발도구를 사용하여 C# 언어로 구현되었습니다. 따라서, .NET 개발환경에서 지원되는 Visual Basic, C++ with managed extentions 등의 다른 언어 개발자들도 이용이 가능합니다.

리더 콘트롤 라이브러리는 리더와의 통신을 위해서 RS-232C 를 지원합니다.

(주)네스랩 “**RFID 동글 또는 모듈형 리더**” 는 통신을 위하여 시리얼 포트 즉, RS-232C 를 제공합니다. 또한, USB-Serial gender 를 통하여 USB 포트에 삽입하여 사용할 수 가 있습니다.

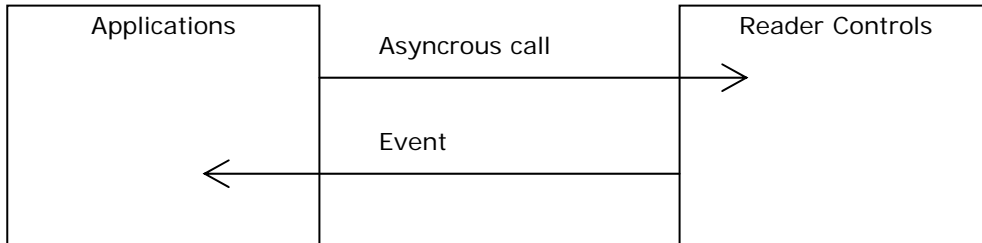
동글형 리더 를 사용하기에 앞서서 별도로 제공된 **USB – Serial 드라이버**를 설치해 줍니다.

The application initiates transactions with ISO 18000-6C tags or tag populations by executing ISO 18000-6C tag-protocol operations. The interface exposes direct access to the following ISO 18000-6C tag-protocol operations:

- Inventory
- Read
- Write
- Kill
- Erase
- Lock

1.3 Architectures

빠른 처리 속도와 원활한 사용자 인터페이스를 제공하기 위하여
비동기 프로그래밍 모델(Asynchronous Programming Model)이 채택되었습니다.



응용 프로그램에서 Reader Control 이 제공하는 기능을 비동기적으로 호출하고
기능 수행이 완료된 직후에 Reader Control 에서 발생하는 이벤트를 처리해 주는 구조입니다.

2. QUICK START

빠른 리더 콘트롤 사용을 위하여 별도의 데모 프로그램을 제공하므로 참조하시기 바랍니다.
제공되는 데모 프로그램은 Visual Studio .NET 개발환경에서 C# 언어로 작성되었습니다.

하기와 같은 순서로 작성하시기 바랍니다.

2.1 리더 콘트롤 참조 추가

Reader Control 을 사용하기 위해서는 프로젝트에 참조 추가 를 해 줍니다.
제공된 "nesslab.reader.api.dll" 파일을 선택하여 참조 추가 를 해주면 하기와 같은 이름의 참조 항목이 추가됩니다.

```
nesslab.reader.api
```

아울러, 하기의 using 지시문을 코드에 추가해 줍니다.

```
using nesslab.reader.api;
```

도구 상자에 추가하여 사용하지 않습니다.

2.2 리더 콘트롤 생성 및 초기화

하기의 데모 코드와 같이 리더 콘트롤 을 생성하고
리더 콘트롤 에서 발생하는 이벤트를 처리하기 위한 이벤트 핸들러를 추가해 줍니다.

```
Reader reader;  
reader = new Reader();  
reader.ReaderEvent += new ReaderEventHandler(OnReaderEvent);  
reader.ModelType = ModelType.NL_RD600;
```

2.3 연결 처리

(주)네스랩 "RFID 동글 또는 모듈형 리더" 는 통신을 위하여 시리얼 포트 즉, RS-232C 를 제공합니다.
또한, USB-Serial gender 를 통하여 USB 포트에 삽입하여 사용할 수 가 있습니다.

동글형 리더 를 사용하기에 앞서서 별도로 제공된 USB – Serial 드라이버를 설치해 줍니다.

```
reader.ConnectSerial("COM5", 115200);
```

상기 함수는 동기적으로 연결을 시작합니다. 연결 수행이 끝나면 하기와 같은 이벤트가 발생합니다.

ReaderEventKind.Connected – 연결에 성공한 경우에 발생합니다.
ReaderEventKind.Disconnected – 연결에 실패한 경우에 발생합니다.

하기 리더 이벤트 핸들러 코드를 참조합니다.

```
private void OnReaderEvent(object sender, ReaderEventArgs e)
{
    //리더 쓰레드에서 호출되므로 메인 쓰레드가 UI 처리를 할 수 있도록 해 줍니다.
    if (this.InvokeRequired)
    {
        this.BeginInvoke(new ReaderEventHandler(OnReaderEvent), new object[] { sender, e });
        return;
    }

    switch (e.Kind)
    {
        //other event...

        case ReaderEventKind.Connected:
            //상태 메시지를 표시한다.
            lblConnectStatus.Text = e.Message;
            break;
        case ReaderEventKind.Disconnected:
        case ReaderEventKind.timeout:
            //상태 메시지를 표시한다.
            lblConnectStatus.Text = e.Message;
            //창닫기를 통해서 발생되었으면 창을 닫아 준다.
            if (e.CloseType == CloseType.FormClose) Close();
            break;
    }
}
```

2.4 연결 해지 및 종료

리더와의 연결 해지 처리 및 자원 회수를 위하여 하기의 함수를 호출해 줍니다.

```
reader.Close(CloseType.Close);
```

리더와의 연결이 해지가 되며

리더와의 명령어 전송 및 수신을 처리하는 리더 쓰레드가 종료되면서 하기와 같은 리더 이벤트가 발생합니다.

```
ReaderEventKind.Disconnected
```

리더와 연결이 된 상태에서 사용자가 창닫기를 하는 경우

하기의 데모 코드와 같이 리더 쓰레드가 정상적으로 종료된 후에 창을 닫아 줄 수 있도록 합니다.

```
protected override void OnClosing(CancelEventArgs e)
{
    //리더 쓰레드가 동작중인가?
    //그렇다면 먼저 리더 쓰레드를 종료해 준다.
    if (reader.IsHandling)
    {
        e.Cancel = true;
        reader.Close(CloseType.FormClose);
    }
    else
    {
        base.OnClosing(e);
    }
}
```

리더와의 연결이 해지되며

리더와의 명령어 전송 및 수신을 처리하는 리더 쓰레드가 종료되면서 하기와 같은 이벤트가 발생합니다.

ReaderEventKind.Disconnected

이벤트 핸들러에서 발생한 이벤트의 CloseType 을 확인하고 창닫기를 처리해 줍니다.

```
case ReaderEventKind.Disconnected:
    //상태 메시지를 표시한다.
    lblConnectStatus.Text = e.Message;
    //창닫기를 통해서 발생되었으면 창을 닫아 준다.
    if (e.CloseType == CloseType.FormClose) Close();
    break;
```

2.5 이벤트 핸들러 작성

리더 컨트롤 에서 발생하는 이벤트들을 처리하는 핸들러를 작성합니다.

이벤트 핸들러는 리더와의 명령어 전송 및 수신을 처리하는 별도의 리더 쓰레드가 호출하는 함수입니다.

별도의 리더 쓰레드에서 호출되므로 메인 쓰레드가 UI 처리를 할 수 있도록 해 줍니다.

하기 데모 코드에 삽입된 코멘트를 참조합니다.

```
private void OnReaderEvent(object sender, ReaderEventArgs e)
{
    //별도의 리더 쓰레드에서 호출되므로 메인 쓰레드가 UI 처리를 할 수 있도록 해 줍니다.
    if (this.InvokeRequired)
    {
        this.BeginInvoke(new ReaderEventHandler(OnReaderEvent), new object[] { sender, e });
        return;
    }

    switch (e.Kind)
    {
        case ReaderEventKind.Connected:
            //상태 메시지를 표시한다.
            lblConnectStatus.Text = e.Message;
            break;
        case ReaderEventKind.Disconnected:
        case ReaderEventKind.timeout:
            //상태 메시지를 표시한다.
            lblConnectStatus.Text = e.Message;
            //창닫기를 통해서 발생되었으면 창을 닫아 준다.
            if (e.CloseType == CloseType.FormClose) Close();
            break;

        case ReaderEventKind.TagId:
        case ReaderEventKind.GetTagMemory:
        case ReaderEventKind.SetTagMemory:
        case ReaderEventKind.LockTag:
        case ReaderEventKind.KillTag:

        case ReaderEventKind.Power:
        case ReaderEventKind.QValue:
        case ReaderEventKind.ScanTime:
        case ReaderEventKind.ContinueMode:
        case ReaderEventKind.Buzzer:
        case ReaderEventKind.Hopping:
        case ReaderEventKind.Session:
            //리더와의 통신시에 사용되는 패킷포맷은 하기와 같으며 - 별도로 제공된 문서참조
            //PACKET FORMAT : [0x7E],PL(1),DATA,CRC(1),[0x7D] - PL(1) = DATA + CRC(1)
            //e.Payload 에는 DATA 가 담겨져 있습니다.
            //제일 처음 바이트에는, 담겨져 있는 데이터의 종류에 대한 코드값이 들어 있습니다.
            byte code = e.Payload[0]; //==DATA
```



```

//바이트 시퀀스를 문자열로 디코딩합니다.
string payload = Encoding.ASCII.GetString(e.Payload, 1, e.Payload.Length - 1);
switch (code)
{
    case 0xA0://Tag Memory
    case 0xA1://PL>255 ?
        //HEX 형태의 TagId 에서 PC 값을 제거하고
        string hex = payload.Substring(4, payload.Length - 4);
        //TEXT 형태로 디코딩한다.
        string txt = reader.MakeTextFromHex(hex);
        lbxDetails.Items.Insert(0, payload + "(" + txt + ")");
        break;
    case 0xAE://Tag Response Code
        //응답코드에 상응하는 문자열을 표시해 줍니다.
        lbxDetails.Items.Insert(0, Reader.Responses(payload));
        break;
    case 0x6E://GetSetting
        lbxDetails.Items.Insert(0, payload);
        break;
}
break;
}
}

```

리더와의 통신시에 사용되는 패킷구조는 하기와 같습니다 – 상세 내용은 별도로 제공된 문서참조

0x7E(1)	PL(1)	DATA	CRC(1)	0x7D(1)
---------	-------	------	--------	---------

리더 컨트롤은 패킷이 수신되면 CRC 확인을 한 후에 **DATA 부분만**(e.Payload) 을 **ReaderEventArgs** 에 포장하여 이벤트를 발생시킵니다.

e.Payload 의 제일 처음 바이트에는 데이터 종류에 대한 코드값이 들어 있습니다.
호출 함수와 상응하는 리더 이벤트를 정리하면 하기 테이블과 같습니다.

Function	Event	Code
ReadTagId	ReaderEventKind.TagId	0xA0, 0xA1
ReadTagMemory	ReaderEventKind.GetTagMemory	0xA0, 0xA1
WriteTagMemory	ReaderEventKind.SetTagMemory	0xAE
LockTag	ReaderEventKind.LockTag	0xAE
KillTag	ReaderEventKind.KillTag	0xAE
GetPower	ReaderEventKind.Power	0x6E
GetQValue	ReaderEventKind.QValue	0x6E
GetContinueMode	ReaderEventKind.ContinueMode	0x6E
GetBuzzer	ReaderEventKind.Buzzer	0x6E
GetScanTime	ReaderEventKind.ScanTime	0x6E
GetHopping	ReaderEventKind.Hopping	0x6E
GetSession	ReaderEventKind.Session	0x6E

2.6 태그 ID 읽기 – Inventory

태그 ID 를 읽기 위하여 하기의 함수를 호출해 줍니다.

```
reader.ReadTagId(ReadType.Multi);
```

```
public enum ReadType
{
    None,
    One, //TAG 가 오직 한 개 존재할 경우에 EPC 코드를 읽기 위해서 사용한다.
    Multi, //여러 개의 TAG 가 존재할 경우 TAG 들이 동시에 응답하게 되어 발생하는 충돌을 방지하기 위해서 사용된다.
}
```

태그가 인식이 되면 하기와 같은 이벤트가 발생되며 이벤트 핸들러에서 처리해 줍니다.

`ReaderEventKind.TagId` – 태그가 인식된 경우 발생합니다.
`ReaderEventKind.timeout` – 리더로부터 응답이 없는 경우에 발생합니다.

RFID 리더로부터 수신된 데이터는 바이트 배열(`e.Payload`)에 들어 있으며, 하기의 함수를 사용하여 문자열로 디코딩해 줍니다.

```
string text = Encoding.ASCII.GetString(e.Payload);
```

RFID 리더로부터 수신된 데이터는 호출 함수에 따라서 태그 ID, 설정값, 응답코드, ... 등이 있을 수 있으므로 제일 처음 바이트의 값에 따라서 처리해 줍니다.

수신된 `TagId` 는 HEX 형태의 포맷인데, 이를 TEXT 형태로 디코딩하려면 하기와 같이 처리해 줍니다.

```
//HEX 형태의 TagId 에서 PC 값을 제거하고
string hex = payload.Substring(4, payload.Length - 4);
//TEXT 형태로 디코딩한다.
string txt = reader.MakeTextFromHex(hex);
```

2.7 동작 중지 – Stop operations

태그 ID 읽기 동작을 중지하기 위하여 하기의 함수를 호출합니다.

```
reader.StopOperation();
```

별도의 이벤트가 발생하지 않습니다.

3. READER API

3.1 CONSTRUCTORS

Reader()

DESCRIPTIONS

Reader 클래스의 새 인스턴스를 초기화 합니다.

PARAMETERS

EXAMPLES

```
Reader reader;  
reader = new Reader();
```

3.2 PROPERTIES

ModelType ModelType

DESCRIPTIONS

리더 모델을 나타내는 값을 설정하거나 가져옵니다.

```
public enum ModelType
{
    None,
    NL_RD600,
}
```

EXAMPLES

```
reader.ModelType = ModelType.NL_RD600;
```

TagType TagType

DESCRIPTIONS

태그 유형을 나타내는 값을 설정하거나 가져옵니다.

```
enum TagType
{
    ISO18000_6B, //지원하지 않습니다.
    ISO18000_6C_GEN2,
}
```

bool IsHandling

DESCRIPTIONS

리더 상태가 Handling 인지를 나타내는 값을 가져옵니다.

리더와 연결이 되고 명령어 전송 및 수신을 처리하는 리더 스레드가 동작중인지를 나타냅니다.

```
enum ReaderState
{
    None = 0x00000000,
    Handling = 0x00000001, //리더 스레드가 동작중인가 ?
}
```

EXAMPLES

```
protected override void OnClosing(CancelEventArgs e)
{
    //리더 스레드가 동작중인가 ?
    //그렇다면 먼저 리더 스레드를 종료해 준다.
    if (reader.IsHandling)
    {
        e.Cancel = true;
        reader.Close(CloseType.FormClose);
    }
    else
    {
        base.OnClosing(e);
    }
}
```

3.3 EVENTS

event ReaderEventHandler ReaderEvent;

DESCRIPTIONS

리더에서 발생되는 이벤트입니다.

EXAMPLES

```
reader.ReaderEvent += new ReaderEventHandler(OnReaderEvent);
```

DETAILED DESCRIPTIONS

```
delegate void ReaderEventHandler(object sender, ReaderEventArgs e);
```

이벤트 발생시 전달되는 정보는 하기 클래스와 같습니다.

```
class ReaderEventArgs : EventArgs
{
    private ReaderEventKind kind;
    private string message;
    private byte[] payload;
    private CloseType closeType = CloseType.None;
}
```

ReaderEventArgs.Message 는 이벤트에 대한 설명을 나타냅니다.

주로, ReaderEventKind.Disconnected 이벤트가 발생했을 때 끊어진 원인을 자세히 표시하고자 할 때 사용됩니다.

ReaderEventArgs.Payload 는 리더로부터 수신받은 바이트 배열형태의 데이터로 문자열로 디코딩하여 사용합니다.

ReaderEventArgs.CloseType 은 하기와 같은 닫기 유형을 나타냅니다.

```
enum CloseType
{
    None, //비정상적으로 연결이 끊어진 경우를 나타냅니다.
    Close, //Close(CloseType.Close) 함수를 호출하는 경우
    FormClose, //Close(CloseType.FormClose) 함수를 호출하는 경우
}
```

Close(CloseType closeType) 함수를 호출하거나 네트워크 오류로 연결이 끊어진 경우에

ReaderEventKind.Disconnected 이벤트가 발생합니다.

이때 CloseType 은 연결이 끊어진 원인을 나타냅니다.

ReaderEventArgs.Kind 은 하기와 같은 이벤트 유형을 나타냅니다.

```
public enum ReaderEventKind
{
    None,

    Connected,
    Disconnected,
    timeout,

    TagId, //get
    GetTagMemory,
    SetTagMemory,
    LockTag,
    KillTag,

    Power,
    QValue,
    ScanTime,
```

```

ContinueMode,
Buzzer,
Hopping,
Session,

Command, //get or set, general command
}

```

3.4 METHODS

static public string Responses(string code)

DESCRIPTIONS

리더로부터 응답코드가 발생할 수 있습니다.
응답코드에 대한 문자열을 가져옵니다.

PARAMETERS

code : 응답코드

RESULTS

리더 이벤트가 발생하지 않습니다.

public bool ConnectSerial(string portName, int baudRate)

DESCRIPTIONS

리더 콘트롤이 초기화되며 동기적으로 연결을 시작합니다.

PARAMETERS

portName : 시리얼 포트명, 예를 들어 "COM1"
baudRate : 보드율, 115200 를 사용합니다.

RESULTS

Thread pool's thread 가 연결요청을 완료하면 하기의 리더 이벤트가 발생합니다.

[ReaderEventKind.Connected](#) – 연결됨

[ReaderEventKind.Disconnected](#) – 연결되지 않음

public void Close(CloseType closeType)

DESCRIPTIONS

리더와의 연결을 해지합니다.

PARAMETERS

closeType : 종료 유형

[enum CloseType](#)

{

[None](#),//비정상적으로 연결이 끊어진 경우를 나타냅니다.

[Close](#),//Close(CloseType.Close) 함수를 호출하는 경우

[FormClose](#),//Close(CloseType.FormClose) 함수를 호출하는 경우

}

RESULTS

하기의 리더 이벤트가 발생된다.

[ReaderEventKind.Disconnected](#)

CloseType 을 확인하고 필요한 경우 처리해 줍니다.

public bool Awake()

DESCRIPTIONS

리더를 동작시킨다.

리더는 파워온, Tag 응답종료, 강제종료 후에 항상 Sleep Mode 로 전환한다. 따라서, 리더를 동작시키기 위해서는 먼저 리더를 WakeUp 시켜 주어야 한다.

하기 예제처럼, 리더 설정 API 를 사용할 시에는 반드시 사용해야 한다.

다른 API 함수들은 내부적으로 처리된다.

RESULTS

true – 성공

false – 실패

EXAMPLES

```

reader.Awake(); //리더 설정 API 를 사용할 시에는 반드시 사용해야 한다.
reader.GetPower();
reader.GetQValue();
reader.GetContinueMode();
reader.GetBuzzer();
reader.GetHopping();
reader.GetSession();
reader.GetScanTime();

reader.SetPower(0);
reader.SetQValue(0);
reader.SetContinueMode(1);
reader.SetBuzzer(1);
reader.SetScanTime(0);
reader.SetHopping(0);
reader.SetSession(0);

```

void ReadTagId(ReadType readType)

DESCRIPTIONS

Inventory 시작 명령어를 전송한다.

PARAMETERS

readType : Inventory 유형을 설정한다.

```

public enum ReadType
{

```

None,

One, //TAG 가 오직 한 개 존재할 경우에 EPC 코드를 읽기 위해서 사용한다.

Multi, //여러 개의 TAG 가 존재할 경우 TAG 들이 동시에 응답하게 되어 발생하는 충돌을 방지하기

위해서 사용된다.

```

}

```

RESULTS

하기의 리더 이벤트가 발생된다.

ReaderEventKind.TagId

ReaderEventKind.timeout – 리더로부터 응답이 없는 경우에 발생된다.

void StopOperation()

DESCRIPTIONS

Inventory 중지 명령어를 전송한다.

RESULTS

리더 이벤트가 발생하지 않는다.

리더로부터 별도의 응답이 없습니다.

void ReadTagMemory(uint bank, uint location, uint length, string password)

DESCRIPTIONS

Tag memory 읽기를 수행하는 명령어를 전송한다.

PARAMETERS

type : 태그 유형

```

enum TagType
{

```

ISO18000_6B, //지원하지 않습니다.

ISO18000_6C_GEN2,

```

}

```

bank : 메모리 뱅크

0 – Reserved

1 – EPC

2 – TID

3 – USER

location : 메모리 위치
단위 : word
length : 길이
password : HEX 형태의 비밀번호
길이 : 4 bytes

RESULTS

리더로부터 응답이 오면 하기의 리더 이벤트가 발생된다.

`ReaderEventKind.GetTagMemory`

`ReaderEventKind.timeout` – 리더로부터 응답이 없는 경우에 발생된다.

EXAMPLES

```
reader.ReadTagMemory(1, 1, 7, null);
reader.ReadTagMemory(1, 1, 1, null);
reader.ReadTagMemory(1, 1, 8, null);
```

void WriteTagMemory(uint bank, uint location, string data, string password)

DESCRIPTIONS

Tag memory 쓰기를 수행하는 명령어를 전송한다.

PARAMETERS

type : 태그 유형

`enum TagType`

```
{
    ISO18000_6B, //지원하지 않습니다.
    ISO18000_6C_GEN2,
}
```

bank : 메모리 뱅크

0 – Reserved

1 – EPC

2 – TID

3 – USER

location : 메모리 위치

단위 : word

data : HEX 형태의 메모리 값

password : HEX 형태의 비밀번호

길이 : 4 bytes

RESULTS

리더로부터 응답이 오면 하기의 리더 이벤트가 발생된다.

`ReaderEventKind.SetTagMemory`

`ReaderEventKind.timeout` – 리더로부터 응답이 없는 경우에 발생된다.

EXAMPLES

```
reader.WriteTagMemory(1, 1, "08001111", null);
reader.WriteTagMemory(1, 1, "3000111122223333444455556666", null);
```

void WriteTagMemory(string tagId, string password)

DESCRIPTIONS

Tag memory 쓰기를 수행하는 명령어를 전송한다.

바이트 배열로 인코딩시에 UTF8 을 사용합니다.

PARAMETERS

tagId : 텍스트 형태의 메모리 값

password : HEX 형태의 비밀번호

길이 : 4 bytes

RESULTS

리더로부터 응답이 오면 하기의 리더 이벤트가 발생된다.

`ReaderEventKind.SetTagMemory`

`ReaderEventKind.timeout` – 리더로부터 응답이 없는 경우에 발생된다.

EXAMPLES

```
reader.WriteTagMemory("대한민국", null);
```

void LockTag(string mask, string action, string password)

DESCRIPTIONS

Tag Lock 을 수행하는 명령어를 전송한다.

PARAMETERS

mask : 마스크값

action : 액션값

password : HEX 형태의 비밀번호

길이 : 4 bytes

RESULTS

리더로부터 응답이 오면 하기의 리더 이벤트가 발생된다.

[ReaderEventKind.LockTag](#)

[ReaderEventKind.timeout](#) – 리더로부터 응답이 없는 경우에 발생된다.

EXAMPLES

```
reader.LockTag("0030", "0020", null);
```

void KillTag(string password)

DESCRIPTIONS

Tag Kill 을 수행하는 명령어를 전송한다.

PARAMETERS

password : HEX 형태의 비밀번호

길이 : 4 bytes

RESULTS

리더로부터 응답이 오면 하기의 리더 이벤트가 발생된다.

[ReaderEventKind.KillTag](#)

[ReaderEventKind.timeout](#) – 리더로부터 응답이 없는 경우에 발생된다.

EXAMPLES

```
reader.KillTag("12345678");
```

void GetPower()

DESCRIPTIONS

Antenna Power 설정값을 읽어 오는 명령어를 전송한다.

RESULTS

리더로부터 응답이 오면 하기의 리더 이벤트가 발생된다.

[ReaderEventKind.Power](#)

[ReaderEventKind.timeout](#) – 리더로부터 응답이 없는 경우에 발생된다.

void GetQValue()

DESCRIPTIONS

Q Value 설정값을 읽어 오는 명령어를 전송한다.

RESULTS

리더로부터 응답이 오면 하기의 리더 이벤트가 발생된다.

[ReaderEventKind.QValue](#)

[ReaderEventKind.timeout](#) – 리더로부터 응답이 없는 경우에 발생된다.

void GetContinueMode()

DESCRIPTIONS

ContinueMode 설정값을 읽어 오는 명령어를 전송한다.

RESULTS

리더로부터 응답이 오면 하기의 리더 이벤트가 발생된다.

[ReaderEventKind.ContinueMode](#)

[ReaderEventKind.timeout](#) – 리더로부터 응답이 없는 경우에 발생된다.

void GetBuzzer()

DESCRIPTIONS

Buzzer 설정값을 읽어 오는 명령어를 전송한다.

RESULTS

리더로부터 응답이 오면 하기의 리더 이벤트가 발생된다.

[ReaderEventKind.Buzzer](#)

[ReaderEventKind.timeout](#) – 리더로부터 응답이 없는 경우에 발생된다.

void GetScanTime()

DESCRIPTIONS

Inventory 실행 시간을 읽어 오는 명령어를 전송한다.

RESULTS

리더로부터 응답이 오면 하기의 리더 이벤트가 발생된다.

[ReaderEventKind.ScanTime](#)

[ReaderEventKind.timeout](#) – 리더로부터 응답이 없는 경우에 발생된다.

void GetHopping()

DESCRIPTIONS

국가별 사용 주파수 범위와 호핑방식을 읽어 오는 명령어를 전송한다.

RESULTS

리더로부터 응답이 오면 하기의 리더 이벤트가 발생된다.

[ReaderEventKind.Hopping](#)

[ReaderEventKind.timeout](#) – 리더로부터 응답이 없는 경우에 발생된다.

void GetSession()

DESCRIPTIONS

Session 값을 읽어 오는 명령어를 전송한다.

RESULTS

리더로부터 응답이 오면 하기의 리더 이벤트가 발생된다.

[ReaderEventKind.Session](#)

[ReaderEventKind.timeout](#) – 리더로부터 응답이 없는 경우에 발생된다.

void SetPower(int value)

DESCRIPTIONS

Antenna power 를 설정하는 명령어를 전송한다.

PARAMETERS

value : 0(full) ~ 31

RESULTS

리더 이벤트가 발생하지 않습니다.

리더로부터 별도의 응답이 없습니다.

void SetQValue(int value)

DESCRIPTIONS

Q Value 을 설정하는 명령어를 전송한다.

PARAMETERS

value : 0 ~ 5

RESULTS

리더 이벤트가 발생하지 않습니다.

리더로부터 별도의 응답이 없습니다.

NOTES

void SetContinueMode(int value)

DESCRIPTIONS

ContinueMode 를 설정하는 명령어를 전송한다.

PARAMETERS

value : 0(Off), 1(On)

RESULTS

리더 이벤트가 발생하지 않습니다.

리더로부터 별도의 응답이 없습니다.

void SetBuzzer(int value)

DESCRIPTIONS

Buzzer 발생여부를 설정하는 명령어를 전송한다.

PARAMETERS

value : 0(Off), 1(On)

RESULTS

리더 이벤트가 발생하지 않습니다.

리더로부터 별도의 응답이 없습니다.

void SetScanTime(int value)

DESCRIPTIONS

Inventory 실행 시간을 설정하는 명령어를 전송한다.

PARAMETERS

value : 0(무한대) ~ 60 sec

RESULTS

리더 이벤트가 발생하지 않습니다.

리더로부터 별도의 응답이 없습니다.

void SetHopping(int value)

DESCRIPTIONS

국가별 사용주파수 범위와 호핑방식을 설정하는 명령어를 전송한다.

PARAMETERS

value : 0 ~ 6

RESULTS

리더 이벤트가 발생하지 않습니다.

리더로부터 별도의 응답이 없습니다.

void SetSession(int value)

DESCRIPTIONS

Session 값을 설정하는 명령어를 전송한다.

PARAMETERS

value : 1, 2

RESULTS

리더 이벤트가 발생하지 않습니다.
리더로부터 별도의 응답이 없습니다.

void SetDefault()

DESCRIPTIONS

리더의 모든 설정값들을 초기화 한다.

PARAMETERS

RESULTS

리더 이벤트가 발생하지 않습니다.
리더로부터 별도의 응답이 없습니다.