**VIKITEK**

**VFR4读写器API函数手册**

目录

[**V3.0** **错误!未定义书签。**](#_Toc434934827)

[1. 引言 13](#_Toc434934828)

[2. API简介 14](#_Toc434934829)

[2.1 API动态库列表 14](#_Toc434934830)

[2.2 创建一个C#工程 14](#_Toc434934831)

[2.3 API调用时序图 15](#_Toc434934832)

[3. 模块初始化 16](#_Toc434934833)

[4. 模块连接关闭 16](#_Toc434934834)

[4.1 连接 16](#_Toc434934835)

[4.1.1 定义 16](#_Toc434934836)

[4.1.2 参数 16](#_Toc434934837)

[4.1.3 返回 16](#_Toc434934838)

[4.1.4 例程 16](#_Toc434934839)

[4.2 断开 17](#_Toc434934840)

[4.2.1 定义 17](#_Toc434934841)

[4.2.2 参数 17](#_Toc434934842)

[4.2.3 返回 17](#_Toc434934843)

[4.2.4 例程 17](#_Toc434934844)

[5. 模块配置 18](#_Toc434934845)

[5.1 天线 18](#_Toc434934846)

[5.1.1 定义 18](#_Toc434934847)

[5.1.2 参数 18](#_Toc434934848)

[5.2 设置天线 18](#_Toc434934849)

[5.2.1 定义 18](#_Toc434934850)

[5.2.2 参数 18](#_Toc434934851)

[5.2.3 例程 19](#_Toc434934852)

[5.3 SearchMode 19](#_Toc434934853)

[5.4 SessionTarget 19](#_Toc434934854)

[5.5 Session 19](#_Toc434934855)

[5.6 标签会话参数 20](#_Toc434934856)

[5.6.1 定义 20](#_Toc434934857)

[5.6.2 参数 20](#_Toc434934858)

[5.7 设置标签会话 20](#_Toc434934859)

[5.7.1 定义 20](#_Toc434934860)

[5.7.2 参数 20](#_Toc434934861)

[5.7.3 例程 21](#_Toc434934862)

[5.8 读取速率 21](#_Toc434934863)

[5.9 设置读取速率 21](#_Toc434934864)

[5.9.1 定义 21](#_Toc434934865)

[5.9.2 参数 21](#_Toc434934866)

[5.9.3 例程 22](#_Toc434934867)

[5.10 工作频段 22](#_Toc434934868)

[5.11 设置工作频段 22](#_Toc434934869)

[5.11.1 定义 22](#_Toc434934870)

[5.11.2 参数 22](#_Toc434934871)

[5.11.3 例程 23](#_Toc434934872)

[5.12 设置盘点时间 23](#_Toc434934873)

[5.12.1 定义 23](#_Toc434934874)

[5.12.2 参数 23](#_Toc434934875)

[5.12.3 例程 23](#_Toc434934876)

[5.13 RSSI过滤参数 24](#_Toc434934877)

[5.13.1 定义 24](#_Toc434934878)

[5.13.2 参数 24](#_Toc434934879)

[5.14 设置RSSI过滤参数 24](#_Toc434934880)

[5.14.1 定义 24](#_Toc434934881)

[5.14.2 参数 24](#_Toc434934882)

[5.14.3 例程 24](#_Toc434934883)

[5.15 GPIO事件 25](#_Toc434934884)

[5.15.1 定义 25](#_Toc434934885)

[5.15.2 参数 26](#_Toc434934886)

[5.16 GPIO 配置 26](#_Toc434934887)

[5.16.1 定义 26](#_Toc434934888)

[5.16.2 参数 26](#_Toc434934889)

[5.16.3 例程 26](#_Toc434934890)

[5.16.4 注意事项 27](#_Toc434934891)

[5.17 模块参数 27](#_Toc434934892)

[5.17.1 定义 27](#_Toc434934893)

[5.17.2 参数 27](#_Toc434934894)

[5.18 获取模块配置 28](#_Toc434934895)

[5.18.1 定义 28](#_Toc434934896)

[5.18.2 参数 28](#_Toc434934897)

[5.18.3 返回 28](#_Toc434934898)

[5.18.4 例程 28](#_Toc434934899)

[5.19 更新模块配置 29](#_Toc434934900)

[5.19.1 定义 29](#_Toc434934901)

[5.19.2 参数 29](#_Toc434934902)

[5.19.3 返回 29](#_Toc434934903)

[5.19.4 例程 29](#_Toc434934904)

[5.20 设置定频 30](#_Toc434934905)

[5.20.1 定义 30](#_Toc434934906)

[5.20.2 参数 30](#_Toc434934907)

[5.20.3 例程 31](#_Toc434934908)

[5.21 测试最优频道参数 31](#_Toc434934909)

[5.21.1 定义 31](#_Toc434934910)

[5.21.2 参数 31](#_Toc434934911)

[5.22 测试最优频道 32](#_Toc434934912)

[5.22.1 定义 32](#_Toc434934913)

[5.22.2 参数 32](#_Toc434934914)

[5.22.3 例程 32](#_Toc434934915)

[5.23 获取最优频道 32](#_Toc434934916)

[5.23.1 定义 32](#_Toc434934917)

[5.23.2 参数 33](#_Toc434934918)

[5.23.3 例程 33](#_Toc434934919)

[5.24 GPO 配置 33](#_Toc434934920)

[5.24.1 定义 33](#_Toc434934921)

[5.24.2 参数 33](#_Toc434934922)

[5.24.3 例程 33](#_Toc434934923)

[5.25 GPI 配置 34](#_Toc434934924)

[5.25.1 定义 34](#_Toc434934925)

[5.25.2 参数 34](#_Toc434934926)

[5.25.3 例程 34](#_Toc434934927)

[5.25.4 注意事项 34](#_Toc434934928)

[5.26 获取模块温度 34](#_Toc434934929)

[5.26.1 定义 34](#_Toc434934930)

[5.26.2 参数 35](#_Toc434934931)

[5.26.3 例程 35](#_Toc434934932)

[5.26.4 注意事项 35](#_Toc434934933)

[5.27 检查天线 35](#_Toc434934934)

[5.27.1 定义 35](#_Toc434934935)

[5.27.2 参数 35](#_Toc434934936)

[5.27.3 例程 35](#_Toc434934937)

[5.28 设置天线驻留时间 36](#_Toc434934938)

[5.28.1 定义 36](#_Toc434934939)

[5.28.2 参数 36](#_Toc434934940)

[5.28.3 例程 36](#_Toc434934941)

[5.29 设置盘点模式 36](#_Toc434934942)

[5.29.1 定义 36](#_Toc434934943)

[5.29.2 参数 37](#_Toc434934944)

[5.29.3 例程 37](#_Toc434934945)

[5.30 获取模块Firmware信息 37](#_Toc434934946)

[5.30.1 定义 37](#_Toc434934947)

[5.30.2 参数 38](#_Toc434934948)

[5.30.3 例程 38](#_Toc434934949)

[6. 标签操作 38](#_Toc434934950)

[6.1 标签响应模式 38](#_Toc434934951)

[6.2 标签定义 38](#_Toc434934952)

[6.2.1 定义 38](#_Toc434934953)

[6.2.2 参数 39](#_Toc434934954)

[6.3 标签响应事件 39](#_Toc434934955)

[6.3.1 定义 39](#_Toc434934956)

[6.3.2 参数 39](#_Toc434934957)

[6.4 盘点回调函数 39](#_Toc434934958)

[6.4.1 定义 39](#_Toc434934959)

[6.4.2 参数 39](#_Toc434934960)

[6.4.3 注意事项 40](#_Toc434934961)

[6.5 内存区域 40](#_Toc434934962)

[6.6 存取参数 40](#_Toc434934963)

[6.6.1 定义 40](#_Toc434934964)

[6.6.2 参数 40](#_Toc434934965)

[6.7 过滤条件参数 41](#_Toc434934966)

[6.7.1 定义 41](#_Toc434934967)

[6.7.2 参数 41](#_Toc434934968)

[6.8 增加过滤条件 42](#_Toc434934969)

[6.8.1 定义 42](#_Toc434934970)

[6.8.2 参数 42](#_Toc434934971)

[6.8.3 返回 42](#_Toc434934972)

[6.8.4 例程 42](#_Toc434934973)

[6.8.5 注意事项 42](#_Toc434934974)

[6.9 移除过滤条件 43](#_Toc434934975)

[6.9.1 定义 43](#_Toc434934976)

[6.9.2 参数 43](#_Toc434934977)

[6.9.3 返回 43](#_Toc434934978)

[6.9.4 例程 43](#_Toc434934979)

[6.10 启动盘点 43](#_Toc434934980)

[6.10.1 定义 43](#_Toc434934981)

[6.10.2 参数 43](#_Toc434934982)

[6.10.3 返回 44](#_Toc434934983)

[6.10.4 例程 44](#_Toc434934984)

[6.11 停止盘点 44](#_Toc434934985)

[6.11.1 定义 44](#_Toc434934986)

[6.11.2 参数 44](#_Toc434934987)

[6.11.3 返回 44](#_Toc434934988)

[6.11.4 例程 44](#_Toc434934989)

[6.12 读 45](#_Toc434934990)

[6.12.1 定义 45](#_Toc434934991)

[6.12.2 参数 45](#_Toc434934992)

[6.12.3 返回 45](#_Toc434934993)

[6.12.4 例程 45](#_Toc434934994)

[6.13 写 45](#_Toc434934995)

[6.13.1 定义 45](#_Toc434934996)

[6.13.2 参数 45](#_Toc434934997)

[6.13.3 返回 45](#_Toc434934998)

[6.13.4 例程 46](#_Toc434934999)

[6.13.5 注意事项 46](#_Toc434935000)

[6.14 锁 46](#_Toc434935001)

[6.14.1 定义 46](#_Toc434935002)

[6.14.2 参数 46](#_Toc434935003)

[6.14.3 返回 46](#_Toc434935004)

[6.14.4 例程 46](#_Toc434935005)

[6.15 解锁 46](#_Toc434935006)

[6.15.1 定义 46](#_Toc434935007)

[6.15.2 参数 47](#_Toc434935008)

[6.15.3 返回 47](#_Toc434935009)

[6.15.4 例程 47](#_Toc434935010)

[6.16 灭活 47](#_Toc434935011)

[6.16.1 定义 47](#_Toc434935012)

[6.16.2 参数 47](#_Toc434935013)

[6.16.3 返回 47](#_Toc434935014)

[6.16.4 例程 48](#_Toc434935015)

[6.17 块写 48](#_Toc434935016)

[6.17.1 定义 48](#_Toc434935017)

[6.17.2 参数 48](#_Toc434935018)

[6.17.3 返回 48](#_Toc434935019)

[6.17.4 例程 48](#_Toc434935020)

[6.17.5 注意事项 48](#_Toc434935021)

[6.18 永久锁区域 48](#_Toc434935022)

[6.19 永久锁 49](#_Toc434935023)

[6.19.1 定义 49](#_Toc434935024)

[6.19.2 参数 49](#_Toc434935025)

[6.19.3 返回 49](#_Toc434935026)

[6.19.4 例程 49](#_Toc434935027)

[6.20 永久解锁 49](#_Toc434935028)

[6.20.1 定义 49](#_Toc434935029)

[6.20.2 参数 50](#_Toc434935030)

[6.20.3 返回 50](#_Toc434935031)

[6.20.4 例程 50](#_Toc434935032)

[6.21 群读 50](#_Toc434935033)

[6.21.1 定义 50](#_Toc434935034)

[6.21.2 参数 50](#_Toc434935035)

[6.21.3 返回 50](#_Toc434935036)

[6.21.4 例程 51](#_Toc434935037)

[6.22 快读 51](#_Toc434935038)

[6.22.1 定义 51](#_Toc434935039)

[6.22.2 参数 51](#_Toc434935040)

[6.22.3 返回 51](#_Toc434935041)

[6.22.4 例程 51](#_Toc434935042)

[6.23 QT 51](#_Toc434935043)

[6.24 设置QT功能 52](#_Toc434935044)

[6.24.1 定义 52](#_Toc434935045)

[6.24.2 参数 52](#_Toc434935046)

[6.24.3 返回 52](#_Toc434935047)

[6.24.4 例程 52](#_Toc434935048)

[7. IP相关操作 53](#_Toc434935049)

[7.1 设置IP地址 53](#_Toc434935050)

[7.1.1 定义 53](#_Toc434935051)

[7.1.2 参数 53](#_Toc434935052)

[7.1.3 返回 53](#_Toc434935053)

[7.1.4 例程 53](#_Toc434935054)

[7.2 设置IP Port 53](#_Toc434935055)

[7.2.1 定义 53](#_Toc434935056)

[7.2.2 参数 53](#_Toc434935057)

[7.2.3 返回 54](#_Toc434935058)

[7.2.4 例程 54](#_Toc434935059)

[7.3 设置子网掩码SubNetMask 54](#_Toc434935060)

[7.3.1 定义 54](#_Toc434935061)

[7.3.2 参数 54](#_Toc434935062)

[7.3.3 返回 54](#_Toc434935063)

[7.3.4 例程 54](#_Toc434935064)

[7.4 设置网关GateWay 55](#_Toc434935065)

[7.4.1 定义 55](#_Toc434935066)

[7.4.2 参数 55](#_Toc434935067)

[7.4.3 返回 55](#_Toc434935068)

[7.4.4 例程 55](#_Toc434935069)

[7.5 设置DHCP 55](#_Toc434935070)

[7.5.1 定义 55](#_Toc434935071)

[7.5.2 参数 56](#_Toc434935072)

[7.5.3 返回 56](#_Toc434935073)

[7.5.4 例程 56](#_Toc434935074)

[7.6 IP配置信息 56](#_Toc434935075)

[7.6.1 定义 56](#_Toc434935076)

[7.6.2 参数 57](#_Toc434935077)

[7.7 获取IP配置信息 57](#_Toc434935078)

[7.7.1 定义 57](#_Toc434935079)

[7.7.2 参数 57](#_Toc434935080)

[7.7.3 返回 58](#_Toc434935081)

[7.7.4 例程 58](#_Toc434935082)

[7.8 更新IP配置信息 58](#_Toc434935083)

[7.8.1 定义 58](#_Toc434935084)

[7.8.2 参数 58](#_Toc434935085)

[7.8.3 返回 58](#_Toc434935086)

[7.8.4 例程 59](#_Toc434935087)

[8. Storage相关操作 59](#_Toc434935088)

[8.1 获取Storage Block数量 59](#_Toc434935089)

[8.1.1 定义 59](#_Toc434935090)

[8.1.2 参数 59](#_Toc434935091)

[8.1.3 返回 59](#_Toc434935092)

[8.1.4 例程 60](#_Toc434935093)

[8.2 设置Storage Block大小 60](#_Toc434935094)

[8.2.1 定义 60](#_Toc434935095)

[8.2.2 参数 60](#_Toc434935096)

[8.2.3 返回 60](#_Toc434935097)

[8.2.4 例程 60](#_Toc434935098)

[8.3 获取Storage Block大小 60](#_Toc434935099)

[8.3.1 定义 60](#_Toc434935100)

[8.3.2 参数 61](#_Toc434935101)

[8.3.3 返回 61](#_Toc434935102)

[8.3.4 例程 61](#_Toc434935103)

[8.4 写 Storage Block 61](#_Toc434935104)

[8.4.1 定义 61](#_Toc434935105)

[8.4.2 参数 61](#_Toc434935106)

[8.4.3 返回 61](#_Toc434935107)

[8.4.4 例程 62](#_Toc434935108)

[8.5 Read Storage Block 62](#_Toc434935109)

[8.5.1 定义 62](#_Toc434935110)

[8.5.2 参数 62](#_Toc434935111)

[8.5.3 返回 62](#_Toc434935112)

[8.5.4 例程 62](#_Toc434935113)

[9. 其它操作 63](#_Toc434935114)

[9.1 产品信息 63](#_Toc434935115)

[9.1.1 定义 63](#_Toc434935116)

[9.1.2 参数 63](#_Toc434935117)

[9.2 获取产品信息 63](#_Toc434935118)

[9.2.1 定义 63](#_Toc434935119)

[9.2.2 参数 64](#_Toc434935120)

[9.2.3 返回 64](#_Toc434935121)

[9.2.4 例程 64](#_Toc434935122)

[9.3 保存配置 64](#_Toc434935123)

[9.3.1 定义 64](#_Toc434935124)

[9.3.2 参数 64](#_Toc434935125)

[9.3.3 返回 64](#_Toc434935126)

[9.3.4 例程 65](#_Toc434935127)

[9.4 设置运行模式 65](#_Toc434935128)

[9.4.1 定义 65](#_Toc434935129)

[9.4.2 参数 65](#_Toc434935130)

[9.4.3 返回 65](#_Toc434935131)

[9.4.4 例程 65](#_Toc434935132)

[9.5 原始响应数据回调函数 66](#_Toc434935133)

[9.5.1 定义 66](#_Toc434935134)

[9.5.2 参数 66](#_Toc434935135)

[9.5.3 返回 66](#_Toc434935136)

[9.5.4 例程 66](#_Toc434935137)

[9.6 打开蜂鸣器 66](#_Toc434935138)

[9.6.1 定义 66](#_Toc434935139)

[9.6.2 参数 66](#_Toc434935140)

[9.6.3 返回 67](#_Toc434935141)

[9.6.4 例程 67](#_Toc434935142)

[10. 附录 67](#_Toc434935143)

[10.1 结果码Result 67](#_Toc434935144)

[10.2 盘点操作例程完整代码 71](#_Toc434935145)

[10.3 读操作例程完整代码 73](#_Toc434935146)

[10.4 写操作例程完整代码 75](#_Toc434935147)

# 引言

本文档适合于使用VIKITEK VFR4四通道超高频RFID读写器进行UHF开发和测试的软件开发人员和软件测试人员。

# API简介

## API动态库列表

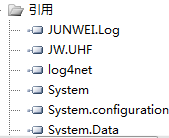
DLL动态库。

JW.UHF.dll c#版本

## 创建一个C#工程

通过VisioStudio 2005以上版本创建一个C#工程，

1. 在工程中引用JW.UHF.dll



## API调用时序图



# 模块初始化

模块初始化通过new一个JWReader对象，所有针对模块的调用均包含在该对象内部。

JWReaderjwReader = new JWReader(“COM3”);

其中”COM3”为模块通过串口线连接PC后PC端对应的串口号。

JWReaderjwReader = new JWReader(“10.10.10.138”,9761);

其中”10.10.10.138”为模块通过网口连接后的IP地址，9761为端口号。

# 模块连接关闭

## 连接

### 定义

public Result RFID\_Open()

建立与模块的连接,连接之前模块必须初始化。

### 参数

无

### 返回

Result.OK

其它Result[参考10.1结果码](#_结果码Result)。

### 例程

Result result = jwReader.RFID\_Open();

if (result != Result.OK)

{

//连接失败

}

//连接成功

## 断开

### 定义

public Result RFID\_Close()

断开与模块的连接

### 参数

无

### 返回

Result.OK

其它Result[参考10.1结果码](#_结果码Result)。

### 例程

Result result =jwReader.RFID\_Close();

if (result != Result.OK)

{

//断开失败

}

//断开成功

# 模块配置

## 天线

### 定义

天线端口

public class AntennaPort

{

public int AntennaIndex;

public int Power;

public bool Exist

}

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| AntennaIndex | 天线端口号 |
| Power | 发射功率 |
| Exist | 只读属性，天线是否存在 |

## 设置天线

### 定义

public Result RFID\_Set\_Antenna(List<AntennaPort> AntennaPortList)

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| List<AntennaPort> AntennaPortList | 天线端口列表,AntennaPort定义[参考5.1.1](#_定义) |

### 例程

//连接模块

List<AntennaPort> antennaPortList = new List<AntennaPort>();

AntennaPort ap = new AntennaPort();

ap.AntennaIndex = 1;//天线1

ap.Power = 27;//功率设为27

antennaPortList.Add(ap);

result = jwReader.RFID\_Set\_Antenna(antennaPortList);

if (result == Result.OK)

Console.WriteLine("Antenna Set Success");

else

Console.WriteLine("Antenna Set Failure");

//断开模块

## SearchMode

读取模式

public enum SearchMode

{

DUAL\_TARGET = 0,// A-B和B-A状态标签均读取

SINGLE\_TARGET = 1,//仅读取A-B状态标签或仅读取B-A状态标签

SINGLE\_TARGET\_WITH\_SUPPRESSION = 2, //仅读取A-B状态标签或仅读取B-A状态标签,只读一次

}

## SessionTarget

目标标签状态

public enum SessionTarget

{

A = 0,//盘点A状态标签

B = 1//盘点B状态标签

}

## Session

会话形式

public enum Session

{

S0 = 0,//标签在被盘点到后立即改变状态 如A状态到B状态后立即转为A状态

S1 = 1,//标签在被盘点到停留0.5-5秒改变状态, 如A状态到B状态停留0.5-5秒后转为A状态

S2 = 2,//标签在被盘点到停留5-60秒改变状态, 如A状态到B状态停留5-60秒后转为A状态

S3 = 3 //与S2相同

}

## 标签会话参数

标签会话参数

### 定义

public class TagGroup

{

public Session Session;

public SessionTarget SessionTarget;

public SearchMode SearchMode;

}

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| SearchMode | 读取模式,参考5.3 |
| SessionTarget | 目标Tag，参考5.4 |
| Session | 会话模式,参考5.5 |

## 设置标签会话

### 定义

public Result RFID\_Set\_Tag\_Group(TagGroup tagGroup)

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| tagGroup | 标签会话参数,[参考5.6](#_定义_1) |

### 例程

//连接模块

#region 设置标签会话参数

TagGroup tg = new TagGroup();

tg.SearchMode = SearchMode.DUAL\_TARGET;

tg.Session = Session.S0;

tg.SessionTarget = SessionTarget.A;

result = jwReader.RFID\_Set\_Tag\_Group(tg);

if (result == Result.OK)

Console.WriteLine("TagGroup Set Success");

else

Console.WriteLine("TagGroup Set Failure");

#endregion

//断开模块

## 读取速率

读取速率

public enum SpeedMode

{

SPEED\_FASTEST = 0,//最快速度读取

SPEED\_NORMAL = 1, //正常速度读取

SPEED\_POWERSAVE = 2//省电模式读取

SPEED\_FULL\_POWER=3 //满电工作 不建议长时间工作状态下使用

}

## 设置读取速率

### 定义

public Result RFID\_Set\_SpeedMode(SpeedMode speedMode)

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| speedMode | 读取速率参数,[参考5.8](#_读取速率) |

### 例程

//连接模块

#region 设置读取速率

SpeedMode sm = SpeedMode.SPEED\_FASTEST;//快速模式

result = jwReader.RFID\_Set\_SpeedMode(sm);

if (result == Result.OK)

Console.WriteLine("Speed Mode Set Success");

else

Console.WriteLine("Speed Mode Set Failure");

#endregion

//断开模块

## 工作频段

工作频段

public enum RegionList

{

FCC = 0,//欧洲频段902-928

CCC = 1,//中国频段920-925

NCC = 2,//台湾频段920-928

OPTIMAL=3 //最优频段

}

## 设置工作频段

### 定义

public Result RFID\_Set\_RegionList(RegionList region)

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| region | 工作频段,[参考5.10](#_工作频段) |

### 例程

//连接模块

#region 设置工作频段

RegionList rl = RegionList.CCC;//CCC频段

result = jwReader.RFID\_Set\_RegionList(rl);

if (result == Result.OK)

Console.WriteLine("RegionList Set Success");

else

Console.WriteLine("RegionList Set Failure");

#endregion

//断开模块

## 设置盘点时间

### 定义

public Result RFID\_Set\_Inventory\_Time(int invTime)

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| invTime | 盘点时间(MS),在盘点invTime 毫秒后就停止。  0代表持续盘点 |

### 例程

//连接模块

#region 设置盘点时间

result = jwReader.RFID\_Set\_Inventory\_Time(1000);//1秒停止盘点

if (result == Result.OK)

Console.WriteLine("Inventory Time Set Success");

else

{

Console.WriteLine("Inventory Time Set Failure");

goto Exit;

}

#endregion

//断开模块

## RSSI过滤参数

### 定义

public class RSSIFilter

{

public bool Enable;

public float RSSIValue;

}

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| Enable | 是否启用过滤 |
| RSSIValue | RSSI值 |

## 设置RSSI过滤参数

### 定义

public Result RFID\_Set\_RSSIFilter(RSSIFilter rssiFilter)

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| rssiFilter | Rssi过滤参数，参考5.17 |

### 例程

//连接模块

#region 设置RSSI 过滤

RSSIFilter rssiFilter=new RSSIFilter();

rssiFilter.Enable =true;

rssiFilter.RSSIValue = (float)-50.7;

result = jwReader.RFID\_Set\_RSSIFilter(rssiFilter);

if (result == Result.OK)

Console.WriteLine("RSSI Filter Set Success");

else

Console.WriteLine("RSSI Filter Set Failure");

#endregion

//断开模块

## GPIO事件

### 定义

public enum GPITriggerValue

{

None=0,//不做动作

Inventory=1,//盘点

Input=2, //输入信号

Hign=3, //将其转变为GPO 高电平输出

Low=4 //将其转变为GPO 低电平输出

}

public enum GPOTriggerValue

{

Low=0, //低电平

Hign=1 //高电平

}

public enum GPIEventType{

Start\_Inventory=0,//启动盘点事件

Stop\_Inventory=1,//停止盘点事件

Hign\_Slow=2,//由高电平到低电平

Slow\_Hign=3 //由低电平到高电平

}

public class GPIEvent

{

public int Port; //GPI 口

public GPIEventType EventType; //GPI事件类型

public GPITriggerValue TriggerValue;//GPI触发器当前值

}

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| Port | GPI 口 |
| EventType | GPI事件类型 |
| TriggerValue | GPI触发器当前值 |

## GPIO 配置

### 定义

public class GPIOConfig

{

public GPITriggerValue GPI0\_VALUE; //GPI0 触发器值

public GPITriggerValue GPI1\_VALUE; //GPI1 触发器值

public GPOTriggerValue GPO0\_VALUE; //GPO0 触发器值

public GPOTriggerValue GPO1\_VALUE; //GPO1 触发器值

}

public Result RFID\_Set\_GPIO(GPIOConfig gpioConfig)

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| gpioConfig | GPIO 配置事件 |

### 例程

//连接模块

#region 设置 GPIO Trigger

GPIOConfig gpioConfig = new GPIOConfig();

gpioConfig.GPI0\_VALUE = GPITriggerValue.Input;

gpioConfig.GPI1\_VALUE = GPITriggerValue.None;

gpioConfig.GPO0\_VALUE = GPOTriggerValue.Hign;

gpioConfig.GPO1\_VALUE = GPOTriggerValue.Hign;

result = jwReader.RFID\_Set\_GPIO(gpioConfig);

#endregion

//断开模块

### 注意事项

必须停止盘点任务

## 模块参数

### 定义

public class RfidSetting

{

public List<AntennaPort> AntennaPort\_List;

public List<GPIOTrigger> GPIO\_Trigger\_List;

public TagGroup Tag\_Group;

public RSSIFilter RSSI\_Filter;

public InventoryMode Inventory\_Mode;

public SpeedMode Speed\_Mode;

public int Inventory\_Time;

public RegionList Region\_List;

public WorkMode Work\_Mode;

}

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| AntennaPort\_List | 天线列表，[AntennaPort定义参考5.1](#_定义) |
| GPIO\_Trigger\_List | GPIO触发器列表,[GPIO触发器参考5.19](#_定义_2) |
| Tag\_Group | 标签会话参数,[参考5.6](#_定义_1) |
| RSSI\_Filter | RSSI过滤参数，[参考5.17](#_定义_3) |
| Inventory\_Mode | 盘点模式，[参考5.29](#_设置盘点模式) |
| Inventory\_Time | 盘点时间(S),盘点多长时间停止，0代表持续盘点 |
| Speed\_Mode | 读取速率,[参考5.8](#_读取速率) |
| Region\_List | 工作频段，[参考5.10](#_工作频段) |
| Work\_Mode | 工作模式,[参考5.12](#_工作模式) |

## 获取模块配置

### 定义

Result RFID\_Get\_Config(RfidSetting setting)

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| setting | 模块配置参数,[参考5.21](#_模块参数) |

### 返回

Result.OK

其它Result[参考10.1结果码](#_结果码Result)。

### 例程

#region 获取模块配置

Console.WriteLine("Get RFID Config");

RfidSetting rs = null;

result = jwReader.RFID\_Get\_Config(out rs);

if (result == Result.OK)

{

foreach( AntennaPort apPort in rs.AntennaPort\_List){

Console.WriteLine(String.Format("AntennaPort={0},Power={1},Exist={2}", apPort.AntennaIndex, apPort.Power, apPort.Exist));

}

foreach (GPIOTrigger gpioTrigger in rs.GPIO\_Trigger\_List){

Console.WriteLine(String.Format("GPIOPort={0},GPIOValue={1}", gpioTrigger.GPIO\_PORT, gpioTrigger.GPIO\_VALUE));

}

Console.WriteLine("SearchMode={0},Session={1},SessionTarget={2}", rs.Tag\_Group.SearchMode, rs.Tag\_Group.Session, rs.Tag\_Group.SessionTarget);

Console.WriteLine("InventoryMode={0},InventoryTime={1}", rs.Inventory\_Mode, rs.Inventory\_Time);

Console.WriteLine("RSSIEnable={0},RSSIValue={1}", rs.RSSI\_Filter.Enable, rs.RSSI\_Filter.RSSIValue);

Console.WriteLine("SpeedMode={0},RegionList={1},WorkMode={2}", rs.Speed\_Mode, rs.Region\_List,rs.Work\_Mode);

}

else

Console.WriteLine("RFID Config Set Failure");

#endregion

## 更新模块配置

### 定义

Result RFID\_Set\_Config(RfidSetting setting)

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| setting | 模块配置参数, [参考5.21](#_模块参数) |

### 返回

Result.OK

其它Result[参考10.1结果码](#_结果码Result)。

### 例程

//连接模块

#region 配置模块

RfidSetting rs=new RfidSetting();

rs.AntennaPort\_List =new List<AntennaPort>();

AntennaPort ap = new AntennaPort();

ap.AntennaIndex = 1;//天线1

ap.Power = 27;//功率设为27

rs.AntennaPort\_List.Add(ap);

rs.GPIO\_Trigger\_List = null;

rs.Inventory\_Time = 10;

rs.Region\_List = RegionList.CCC;

rs.RSSI\_Filter = new RSSIFilter();

rs.RSSI\_Filter.Enable = true;

rs.RSSI\_Filter.RSSIValue = (float)-50.7;

rs.Speed\_Mode = SpeedMode.SPEED\_FASTEST;

rs.Tag\_Group = new TagGroup();

rs.Tag\_Group.SessionTarget = SessionTarget.A;

rs.Tag\_Group.SearchMode = SearchMode.DUAL\_TARGET;

rs.Tag\_Group.Session = Session.S0;

result = jwReader.RFID\_Set\_Config(rs);

if (result == Result.OK)

Console.WriteLine("RFID Config Set Success");

else

Console.WriteLine("RFID Config Set Failure");

#endregion

//断开连接

## 设置定频

### 定义

public Result RFID\_Set\_Fix\_Frequency(double frequency)//仅支持902-928频段的频率

public Result RFID\_Set\_Fix\_Frequency(int port, int power, double frequency)//支持840-960任意频率

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| port | 天线口 |
| power | 天线输出功率0-30 |
| frequency | 频率值 如922.25 922.75等 |

### 例程

//连接模块

#region 设置定频;

result = jwReader.RFID\_Set\_Fix\_Frequency(922.25);

if (result == Result.OK)

Console.WriteLine("Frequency Set Success");

else

Console.WriteLine("Frequency Set Failure");

#endregion

//断开模块

## 测试最优频道参数

### 定义

/// <summary>

/// 频道性能

/// </summary>

public class ChannelPerformance

{

/// <summary>

/// 天线端口

/// </summary>

public int Antenna\_Port;

/// <summary>

/// 天线功率

/// </summary>

public int Antenna\_Power;

/// <summary>

/// 反向功率值

/// </summary>

public int Reverse\_Power\_Value;

}

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| Antenna\_Port | 天线口 |
| Antenna\_Power | 天线功率 |
| Reverse\_Power\_Value | 反向功率限定值 |

## 测试最优频道

### 定义

public Result RFID\_Test\_Channel(ChannelPerformance cpf,out List<int> channelList)

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| cpf | 最优频道参数，[参考5.21](#_测试最优频道参数) |
| channelList | 最优频道列表 |

### 例程

//连接模块

List<int> channelList = null;

ChannelPerformance cpf = new ChannelPerformance();

cpf.Antenna\_Port = 0;

cpf.Antenna\_Power = 30;

cpf.Reverse\_Power\_Value = 10;

Result result = jwReader.RFID\_Test\_Channel(cpf,out channelList);

if (result == Result.OK)//测试完成

{

}

//断开模块

## 获取最优频道

### 定义

public Result RFID\_Get\_Optimal\_Channel(out List<int> channelList)

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| channelList | 最优频道列表 |

### 例程

//连接模块

List<int> channelList = null;

Result result = jwReader.RFID\_Get\_Optimal\_Channel(out channelList);;

if (result == Result.OK)//最优频道列表获取完成

{

}

//断开模块

## GPO 配置

### 定义

public Result RFID\_Set\_GPO(GPIOConfig gpioConfig)

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| gpioConfig | GPIO 配置事件 |

### 例程

//连接模块

#region 设置 GPO Trigger

GPIOConfig gpioConfig = new GPIOConfig();

gpioConfig.GPO0\_VALUE = GPOTriggerValue.Hign;

gpioConfig.GPO1\_VALUE = GPOTriggerValue.Hign;

result = jwReader.RFID\_Set\_GPO(gpioConfig);

#endregion

//断开模块

## GPI 配置

### 定义

public Result RFID\_Set\_GPI(GPIOConfig gpioConfig)

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| gpioConfig | GPIO 配置事件 |

### 例程

//连接模块

#region 设置 GPI Trigger

GPIOConfig gpioConfig = new GPIOConfig();

gpioConfig.GPI0\_VALUE = GPITriggerValue.Input;

gpioConfig.GPI1\_VALUE = GPITriggerValue.None;

result = jwReader.RFID\_Set\_GPI(gpioConfig);

#endregion

//断开模块

### 注意事项

必须停止盘点任务

## 获取模块温度

### 定义

public Result RFID\_Get\_Temperature(out int temperature)

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| temperature | 模块温度 |

### 例程

//连接模块

#region 获取模块温度

int temperature=0;

result = jwReader.RFID\_Get\_Temperature(out temperature);

#endregion

//断开模块

### 注意事项

必须停止盘点任务

## 检查天线

### 定义

public Result RFID\_Check\_Antenna(int antennaIndex)

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| antennaIndex | 天线索引 |

### 例程

//连接模块

#region 检查天线0状况

result = jwReader.RFID\_Check\_Antenna(0);

#endregion

//断开模块

## 设置天线驻留时间

### 定义

public Result RFID\_Set\_DWellTime(int dwellTime)

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| dwellTime | 天线驻留时间(Ms) |

### 例程

//连接模块

#region 设置天线驻留300ms

result = jwReader. RFID\_Set\_DWellTime(300);

#endregion

//断开模块

## 设置盘点模式

### 定义

/// <summary>

/// 盘点模式

/// </summary>

public enum InventoryMode

{

/// <summary>

/// 持续盘点

/// </summary>

Continue = 0,//持续盘点

/// <summary>

/// 仅一个

/// </summary>

OnlyOne = 1//盘点到一个标签即停止

}

public Result RFID\_Set\_Inventory\_Mode(InventoryMode mode)

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| mode | 盘点模式 |

### 例程

//连接模块

#region 设置读写器为持续盘点模式

result = jwReader.RFID\_Set\_Inventory\_Mode(InventoryMode.Continue);

#endregion

//断开模块

## 获取模块Firmware信息

### 定义

/// <summary>

/// 分位信息

/// </summary>

public class FirmwareInfo

{

/// <summary>

/// 主版本号

/// </summary>

public int Main\_Version//主版本号

/// <summary>

/// 副版本号

/// </summary>

public int Sub\_Version //副版本号

}

public Result RFID\_Get\_Firmware\_Info(out FirmwareInfo firmwareInfo)

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| firmwareInfo | 分位信息 |

### 例程

//连接模块

#region 设置读写器为持续盘点模式

FirmwareInfo firmwareInfo = new FirmwareInfo();

result = jwReader. RFID\_Get\_Firmware\_Info(out firmwareInfo);

#endregion

//断开模块

# 标签操作

## 标签响应模式

标签响应模式

public enum TagMode

{

More=0,//一次盘点可以响应多次

One=1 //一次盘点仅响应一次 目前仅支持EPC 96bit 1000个标签以内。

}

## 标签定义

### 定义

public class Tag

{

public String DATA;

public String EPC;

public float RSSI;

public int PORT;

}

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| DATA | 读到数据（快读FAST\_READ返回） |
| EPC | 当前EPC值 |
| PORT | 天线端口号 |
| RSSI | 标签返回的RSSI值 |

## 标签响应事件

### 定义

public class TagsEventArgs:EventArgs

{

public Tag tag;//标签信息

public String errorCode;//错误码

}

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| tag | 当前Tag,[参考6.2](#_定义_4) |

## 盘点回调函数

### 定义

public delegate void TagsEventHandler(object sender, TagsEventArgs args)

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| sender | 上下文句柄对象 |
| args | 标签响应事件，[参考6.3](#_定义_5) |

### 注意事项

回调函数内部不能包含耗时操作，如更新界面，写数据库等。所有耗时操作请通过其它线程完成

## 内存区域

内存区域

public enum MemoryBank

{

RESERVED = 0,// RESERVED区

EPC = 1,//EPC区

TID = 2,//TID区

USER = 3//USER区

}

## 存取参数

### 定义

public class AccessParam

{

public string AccessPassword;

public MemoryBank Bank;

public int OffSet;

public int Count;

}

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| Bank | 内存区域 |
| Count | 读取字节数（Byte）以偶数字节数(Word)形式读 |
| AccessPassword | 存取密码 |
| Offset | 偏移量（Byte） |

## 过滤条件参数

### 定义

public class RfidCriteria

{

public MemoryBank Bank;

public int Count;

public byte[] Mask;

public int Offset;

public bool Match;

}

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| Bank | 内存区域，[参考6.5](#_内存区域),其中的RESERVED不是有效值 |
| Count | 匹配字节数（Byte）0到31之间 |
| Offset | 偏移量(Byte) |
| Mask | 匹配数据  举例：如果你想获取从EPC值第2个Byte开始的2个Byte值为”ABCD”的标签。则  对应值为:  bank = MemoryBank.EPC  offset = 2(前面32bit为长度等信息)  count = 2  mask[0] = 0xAB  mask[1] = 0xCD |
| Match | 选择匹配的数据还是不匹配的数据  True :选择匹配数据  False:选择不匹配数据 |

## 增加过滤条件

### 定义

public Result RFID\_Set\_Criteria(RfidCriteria criteria)

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| criteria | 过滤条件参数，参考6.7 |

### 返回

Result.OK

其它Result[参考10.1结果码](#_结果码Result)。

### 例程

//连接模块

#region 设置Criteria

RfidCriteria criteria = new RfidCriteria();

criteria.Bank = MemoryBank.EPC;

criteria.OffSet = 0;

criteria.Mask = Util.ToHexByte("3008");

criteria.Count = 2;

criteria.Match = true;

result=jwReader.RFID\_Set\_Criteria(criteria);

if (result == Result.OK)

Console.WriteLine("Set Criteria Success");

else

Console.WriteLine("Set Criteria Failure");

#endregion

//断开连接

以上代码作用为仅选取EPC值从0开始，以”3008”(16进制)开头的标签。

### 注意事项

过滤条件全局有效，包括盘点，读写锁灭活等。

## 移除过滤条件

### 定义

public Result RFID\_Clear\_Criteria()

### 参数

无

### 返回

Result.OK

其它Result[参考10.1结果码](#_结果码Result)。

### 例程

//连接模块

jwReader.RFID\_Clear\_Criteria()

//断开连接

## 启动盘点

### 定义

public Result RFID\_Start\_Inventory()

public void RFID\_Start\_Inventory(bool async)

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| async | 是否异步盘点  True:异步  False:同步(默认) |

### 返回

同步模式下有返回值：

Result.OK

其它Result[参考10.1结果码](#_结果码Result)。

异步模式无返回值

### 例程

[参考附录10.2](#_盘点操作例程完整代码)

## 停止盘点

### 定义

public Result RFID\_Stop\_Inventory()

终止盘点

### 参数

无

### 返回

Result.OK

其它Result[参考10.1结果码](#_结果码Result)。

### 例程

//连接模块

jwReader.RFID\_Stop\_Inventory();

//断开连接

## 读

### 定义

public Result RFID\_Read(AccessParam ap);

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| ap | [存取参数,参考6.6](#_存取参数)，最大支持128 Bytes |

### 返回

Result.OK

其它Result参考10.1结果码。

### 例程

[参考附录10.2](#_读操作例程完整代码)

## 写

### 定义

public Result RFID\_Write(AccessParam ap, ushort[] writedata

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| ap | 存取参数,参考6.6 |
| writedata | 写入数据最大支持128 Bytes |

### 返回

Result.OK

其它Result[参考10.1结果码](#_结果码Result)。

### 例程

[参考附录10.3](#_写操作例程完整代码)

### 注意事项

写模式下,修改EPC内容同时会修改EPC长度。

## 锁

### 定义

public Result RFID\_Lock(string lockPwd)

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| lockPwd | 锁密码 |

### 返回

Result.OK

其它Result[参考10.1结果码](#_结果码Result)。

### 例程

//连接模块

jwReader.RFID\_Lock(0x11111111);

//断开连接

## 解锁

### 定义

public Result RFID\_UnLock(string lockPwd);

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| lockPwd | 解锁密码 |

### 返回

Result.OK

其它Result[参考10.1结果码](#_结果码Result)。

### 例程

//连接模块

jwReader.RFID\_Unlock (0x11111111);

//断开连接

## 灭活

### 定义

public Result RFID\_Kill(string accessPwd, string killPwd);

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| accessPwd | 存取密码 |
| killPwd | 灭活密码 |

### 返回

Result.OK

其它Result[参考10.1结果码](#_结果码Result)。

### 例程

//连接模块

jwReader.RFID\_Kill(0x11111111,0x11111111);

//断开连接

## 块写

### 定义

public Result RFID\_BlockWrite(AccessParam ap, ushort[] writedata

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| ap | 存取参数,参考6.6 |
| writedata | 写入数据 最大支持128Byte |

### 返回

Result.OK

其它Result[参考10.1结果码](#_结果码Result)。

### 例程

[参考附录10.3](#_写操作例程完整代码)

### 注意事项

块写模式下,仅修改EPC内容不修改EPC长度。

## 永久锁区域

public enum LockMemory

{

EPC = 0,

User = 1,

All = 2

}

## 永久锁

### 定义

public Result RFID\_PermLock(LockMemory lockMemory)

标签必须处于解锁状态下，才可操作

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| lockMemory | 永久锁区域 |

### 返回

Result.OK

其它Result[参考10.1结果码](#_结果码Result)。

### 例程

//连接模块

jwReader.RFID\_PermLock(LockMemory.All);

//断开连接

## 永久解锁

### 定义

public Result RFID\_PermUnLock(LockMemory lockMemory);

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| lockMemory | 永久解锁区域 |

### 返回

Result.OK

其它Result[参考10.1结果码](#_结果码Result)。

### 例程

//连接模块

jwReader.RFID\_PermUnLock(LockMemory.All);

//断开连接

## 群读

### 定义

public Result RFID\_GroupRead (AccessParam ap)

public Result RFID\_GroupRead (AccessParam ap ,bool async)

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| ap | [存取参数,参考6.6](#_存取参数) |
| async | 是否异步群读  True:异步  False:同步(默认) |

### 返回

同步模式下有返回值：

Result.OK

其它Result[参考10.1结果码](#_结果码Result)。

异步模式无返回值

### 例程

[参考附录10.2](#_盘点操作例程完整代码)盘点例程

## 快读

### 定义

public Result RFID\_Fast\_Read(AccessParam ap, out Tag tag);

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| ap | [存取参数,参考6.6](#_存取参数) |
| tag | 返回标签数据 |

### 返回

Result.OK

其它Result参考10.1结果码。

### 例程

[参考附录10.2](#_读操作例程完整代码)读例程

## QT

public enum QT

{

/// <summary>

/// 公有区

/// </summary>

PUBLIC=0,

/// <summary>

/// 私有区

/// </summary>

PRIVATE=1

}

## 设置QT功能

### 定义

public Result RFID\_Set\_QT(QT qt);

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| qt | [QT参数,参考6.23](#_QT) |

### 返回

Result.OK

其它Result参考10.1结果码。

### 例程

#region 设置QT Public

Console.WriteLine("Set QT Public");

result = jwReader.RFID\_Set\_QT(QT.PUBLIC);

if (result == Result.OK)

Console.WriteLine("Set QT Public Success");

else

{

Console.WriteLine("Set QT Public Failure");

}

#endregion

# IP相关操作

## 设置IP地址

### 定义

public Result IP\_Set\_Address(string ipAddress)

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| ipAddress | Ip地址 如10.10.10.101 |

### 返回

Result.OK

其它Result[参考10.1结果码](#_结果码Result)。

### 例程

//连接模块

jwReader.IP\_Set\_Address(“10.10.10.101”);

//断开连接

注意事项:

通过9.3 保存UHF和IP相关信息 重启生效

## 设置IP Port

### 定义

public Result IP\_Set\_Port(int port)

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| port | 网口端口 如9761 |

### 返回

Result.OK

其它Result[参考10.1结果码](#_结果码Result)。

### 例程

//连接模块

jwReader.IP\_Set\_Port (9761);

//断开连接

注意事项:

通过9.3 保存UHF和IP相关信息 重启生效

## 设置子网掩码SubNetMask

### 定义

public Result IP\_Set\_SubNet\_Mask(string ipAddress)

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| ipAddress | Ip地址 如255.255.255.0 |

### 返回

Result.OK

其它Result[参考10.1结果码](#_结果码Result)。

### 例程

//连接模块

jwReader.IP\_Set\_SubNet\_Mask(“255.255.255.0”);

//断开连接

注意事项:

通过9.3 保存UHF和IP相关信息 重启生效

## 设置网关GateWay

### 定义

public Result IP\_Set\_GateWay(string ipAddress)

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| ipAddress | Ip地址 如10.10.10.1 |

### 返回

Result.OK

其它Result[参考10.1结果码](#_结果码Result)。

### 例程

//连接模块

jwReader.IP\_Set\_GateWay(“10.10.10.1”);

//断开连接

注意事项:

通过9.3 保存UHF和IP相关信息 重启生效

## 设置DHCP

### 定义

public Result IP\_Set\_DHCP(bool enable)

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| enable | 启用DHCP还是禁用DHCP  True Or False |

### 返回

Result.OK

其它Result[参考10.1结果码](#_结果码Result)。

### 例程

//连接模块

jwReader.IP\_Set\_DHCP (true);

//断开连接

注意事项:

通过9.3 保存UHF和IP相关信息 重启生效

## IP配置信息

### 定义

public class IPConfiguration

{

/// <summary>

/// 是否启用DHCP

/// </summary>

public bool Dhcp;

/// <summary>

/// IP地址

/// </summary>

public string IP;

/// <summary>

/// 端口

/// </summary>

public int Port;

/// <summary>

/// 子网掩码

/// </summary>

public string SubNet\_Mask;

/// <summary>

/// 网关

/// </summary>

public string Gateway;

}

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| Dhcp | 是否启用DHCP |
| IP | IP地址 |
| Port | 天线端口 |
| SubNet\_Mask | 子网掩码 |
| Gateway | 网关 |

注意事项:

通过9.3 保存UHF和IP相关信息 重启生效

## 获取IP配置信息

### 定义

public Result IP\_Get\_Configuration(out IPConfiguration ipConfiguration)

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| ipConfiguration | IP配置信息 |

### 返回

Result.OK

其它Result[参考10.1结果码](#_结果码Result)。

### 例程

//连接模块

IPConfiguration ip;

Result result = jwReader.IP\_Get\_Configuration(out ip);

if (result == Result.OK)

{

//ip.IP;

//ip.Port.ToString();

//ip.SubNet\_Mask;

//ip.Gateway;

//ip.Dhcp;

}

//断开连接

注意事项:

通过9.3 保存UHF和IP相关信息 重启生效

## 更新IP配置信息

### 定义

public Result IP\_Set\_Configuration(IPConfiguration ipConfiguration)

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| ipConfiguration | IP配置信息 |

### 返回

Result.OK

其它Result[参考10.1结果码](#_结果码Result)。

### 例程

//连接模块

IPConfiguration ip = new IPConfiguration();

ip.IP = “10.10.10.121”;

ip.Port = 9761;

ip.SubNet\_Mask = “255.255.255.0”;

ip.Gateway = “10.10.10.1”;

ip.Dhcp = true;

Result result = jwReader.IP\_Set\_Configuration(ip);

//断开连接

注意事项:

通过9.3 保存UHF和IP相关信息 重启生效

# Storage相关操作

## 获取Storage Block数量

### 定义

public Result Storage\_Get\_Block\_Count(out UInt16 count)

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| ipAddress | Ip地址 如10.10.10.101 |

### 返回

Result.OK

其它Result[参考10.1结果码](#_结果码Result)。

### 例程

//连接模块

ushort count=0;

jwReader.Storage\_Get\_Block\_Count (out count);

//断开连接

## 设置Storage Block大小

### 定义

public Result Storage\_Set\_Block\_Size(byte size)

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| size | 设置Block大小，最大不超过0x10 |

### 返回

Result.OK

其它Result[参考10.1结果码](#_结果码Result)。

### 例程

//连接模块

byte size=0x10;

jwReader.Storage\_Set\_Block\_Size (size);

//断开连接

## 获取Storage Block大小

### 定义

public Result Storage\_Get\_Block\_Size(out byte size)

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| size | Block大小，最大不超过0x10 |

### 返回

Result.OK

其它Result[参考10.1结果码](#_结果码Result)。

### 例程

//连接模块

byte size;

jwReader.Storage\_Get\_Block\_Size (out size);

//断开连接

## 写 Storage Block

### 定义

public Result Storage\_Write\_Block\_Data(UInt16 blockIndex, string writeData)

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| blockIndex | block位置,当前blockIndex不能超过8.1 获取的Block数量并且最大不超过1000 |
| writeData | 写入数据,长度必需为8.3 获取的Block大小。 |

### 返回

Result.OK

其它Result[参考10.1结果码](#_结果码Result)。

### 例程

//连接模块

//比如Block Size为0x0C 长度12

Result result = jwReader.Storage\_Write\_Block\_Data(0x01, “000102030405060708090A0B”);

//断开连接

## Read Storage Block

### 定义

public Result Storage\_Read\_Block\_Data(UInt16 blockIndex,out string readData)

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| blockIndex | block位置,当前blockIndex不能超过8.1 获取的Block数量并且最大不超过1000 |
| readData | 读出数据。 |

### 返回

Result.OK

其它Result[参考10.1结果码](#_结果码Result)。

### 例程

//连接模块

string readData;

Result result = jwReader.Storage\_Read\_Block\_Data(0x01, out readData);

//断开连接

# 其它操作

## 产品信息

### 定义

public class ProductInfo

{

public string COMPANY\_NO;//公司编号

public string PRODUCT\_DATE;//生产日期

public string MODEL\_TYPE;//模块型号

public int MODEL\_VERSION;//模块版本

public string MODEL\_SEQUENCE\_NUMBER;//模块序列号

public int ANTENNA\_NUMBER;//天线个数

public List<AntennaPort> ANTENNA\_PORT\_EXIST\_LIST; //天线存在列表

public ProductType PRODUCT\_TYPE;

}

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| ANTENNA\_NUMBER | 天线个数 |
| ANTENNA\_PORT\_EXIST\_LIST | 描述天线是否可用。利用AntennaPort的Exist属性 |
| COMPANY\_NO | 公司编号 |
| PRODUCT\_DATE | 生产日期 |
| MODEL\_TYPE | 模块类型 |
| MODEL\_VERSION | 模块版本 |
| MODEL\_SEQUENCE\_NUMBER | 模块序列号 |
| PRODUCT\_TYPE | 产品类型 |

## 获取产品信息

### 定义

public Result RFID\_Get\_Product\_Info(out ProductInfo productInfo)

获取当前连接模块的相关信息，序列号，天线个数，模块类型等。

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| productInfo | 产品信息类，[参考7.1](#_产品信息) |

### 返回

Result.OK

其它Result[参考10.1结果码](#_结果码Result)

### 例程

//连接模块

Module\_Info info = new Module\_Info();

Result result = jwReader.RFID\_Get\_Product\_Info(ref info);

//断开连接

## 保存配置

### 定义

public Result Save\_Config()

保存UHF和IP等相关信息。

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| productInfo | 产品信息类，[参考7.1](#_产品信息) |

### 返回

Result.OK

其它Result[参考10.1结果码](#_结果码Result)

### 例程

//连接模块

Result result = jwReader.Save\_Config();

//断开连接

## 设置运行模式

### 定义

public enum RunningMode

{

/// <summary>

/// API模式 数据会以API解析后形式返回

/// </summary>

API=0,

/// <summary>

/// 命令模式 原始数据返回

/// </summary>

COMMAND=1

}

public void Set\_Running\_Mode(RunningMode \_runMode)

设置运行模式

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| \_runMode | 运行模式 |

### 返回

### 例程

//连接模块

jwReader.Set\_Running\_Mode(RunningMode.COMMAND);

//断开连接

## 原始响应数据回调函数

### 定义

public delegate void CommandResponseEventHandler(JWReader reader, byte[] responseData);

### 参数

|  |  |
| --- | --- |
| 参数 | 描述 |
| reader | 读写器对象 |
| responseData | 响应数据 |

### 返回

### 例程

//连接模块

jwReader.commandResponseEventReported += CommandResponseEventReport;

//断开模块

private void CommandResponseEventReport(object sender, byte[] data)

{

//处理响应数据

}

## 打开蜂鸣器

### 定义

public Result Open\_Buzzer();

### 参数

### 返回

### 例程

//连接模块

jwReader.Open\_Buzzer();//蜂鸣器响一声

//断开模块

# 附录

## 结果码Result

public enum Result

{

/// <summary>

/// OK

/// </summary>

OK = 0,

/// <summary>

/// 设备忙

/// </summary>

Module\_Is\_Busy = 1,//模块Busy

/// <summary>

/// 设备已经关闭

/// </summary>

Module\_Is\_Closed = 2,//模块是关闭的

/// <summary>

/// 设备已经打开

/// </summary>

Module\_Is\_Already\_Opened = 3,//模块已经打开

/// <summary>

/// 设备无响应

/// </summary>

Model\_Not\_Response = 4,//模块无响应

/// <summary>

/// 发送指令失败

/// </summary>

Send\_Instruct\_Failure = 5,

/// <summary>

/// 响应失败

/// </summary>

Response\_TimeOut = 6,

/// <summary>

/// 连接失败

/// </summary>

Connect\_Failure = 7,

/// <summary>

/// 断开连接失败

/// </summary>

Disconnect\_Failure = 8,

/// <summary>

/// 反响功率过高

/// </summary>

Reverse\_Power\_Too\_Hign = 9,

/// <summary>

/// SDK忙

/// </summary>

Device\_Is\_Busy = 10,

/// <summary>

/// 写数据失败

/// </summary>

Write\_Data\_Is\_Null = 11,

/// <summary>

/// 密码是空的

/// </summary>

Pwd\_Is\_Null = 12,

/// <summary>

/// 密码长度错误

/// </summary>

Pwd\_Length\_Is\_Error = 13,

/// <summary>

/// 天线不存在

/// </summary>

Antenna\_Not\_Exists = 14,

/// <summary>

/// SDK等待超时

/// </summary>

Wait\_TimeOut = 15,

/// <summary>

/// 读到数据为空

/// </summary>

Read\_Data\_Is\_Empty = 16,

/// <summary>

/// SDK连接超时

/// </summary>

Sdk\_Connect\_TimeOut=17,

/// <summary>

/// 接口不可用

/// </summary>

Interface\_Not\_Avaliable=18,

/// <summary>

/// 天线未配置

/// </summary>

Antenna\_Not\_Configure=19,

/// <summary>

/// 频道不可用

/// </summary>

Channel\_Not\_Supported = 20,

/// <summary>

/// 写数据太长

/// </summary>

Write\_Data\_Too\_Long=21,

/// <summary>

/// 网络异常

/// </summary>

Network\_Exception=22,

/// <summary>

/// 接收缓冲区溢出

/// </summary>

Receive\_Buffer\_OverFlow=23,

/// <summary>

/// 解析盘点包错误

/// </summary>

Parse\_Packet\_Data\_Error=24,

/// <summary>

/// 串口异常

/// </summary>

Serial\_Exception=25,

/// <summary>

/// 天线不存在

/// </summary>

Antenna\_Not\_Connected = 26,

/// <summary>

/// 前向功率不足

/// </summary>

Forward\_Power\_InSufficient=27,

/// <summary>

/// 未知异常

/// </summary>

Unknown\_Exception = 99

}

## 盘点操作例程完整代码

using System;

using System.Collections.Generic;

using System.Text;

using JW.UHF;

namespace Inventory

{

class Program

{

/// <summary>

/// 数据上报

/// </summary>

/// <param name="sender"></param>

/// <param name="args"></param>

private static void TagsReport(object sender, TagsEventArgs args)

{

Tag tag=args.tag;

Console.WriteLine(string.Format("EPC={0},Port={1},RSSI={2}", tag.EPC, tag.PORT, tag.RSSI));

}

static void Main(string[] args)

{

JWReader jwReader = new JWReader("COM3");

//JWReader jwReader = new JWReader("10.10.10.121",9761);

#region 打开模块

Result result = jwReader.RFID\_Open();

if (result == Result.OK)

Console.WriteLine("Open Module Success");

else

{

Console.WriteLine("Open Module Failure");

goto Exit;

}

#endregion

#region 配置模块

RfidSetting rs = new RfidSetting();

rs.AntennaPort\_List = new List<AntennaPort>();

AntennaPort ap = new AntennaPort();

ap.AntennaIndex = 1;//天线1

ap.Power = 22;//功率设为27

rs.AntennaPort\_List.Add(ap);

rs.GPIO\_Trigger\_List = null;

rs.Inventory\_Time = 5000;

rs.Region\_List = RegionList.CCC;

rs.RSSI\_Filter = new RSSIFilter();

rs.RSSI\_Filter.Enable = true;

rs.RSSI\_Filter.RSSIValue = (float)-70;

rs.Speed\_Mode = SpeedMode.SPEED\_FASTEST;

rs.Tag\_Group = new TagGroup();

rs.Tag\_Group.SessionTarget = SessionTarget.A;

rs.Tag\_Group.SearchMode = SearchMode.DUAL\_TARGET;

rs.Tag\_Group.Session = Session.S0;

result = jwReader.RFID\_Set\_Config(rs);

if (result == Result.OK)

Console.WriteLine("RFID Config Set Success");

else

{

Console.WriteLine("RFID Config Set Failure");

goto Exit;

}

#endregion

#region 盘点

Console.WriteLine("Start Inventory");

jwReader.TagsReported += TagsReport;

jwReader.RFID\_Start\_Inventory();

Console.WriteLine("Stop Inventory");

#endregion

Exit:

#region 关闭模块

result = jwReader.RFID\_Close();

if (result == Result.OK)

Console.WriteLine("Close Module Success");

else

{

Console.WriteLine("Close Module Failure");

}

#endregion

Console.ReadLine();

}

}

}

## 读操作例程完整代码

using System;

using System.Collections.Generic;

using System.Text;

using JW.UHF;

namespace Read

{

class Program

{

static void Main(string[] args)

{

JWReader jwReader = new JWReader("COM3");

//JWReader jwReader = new JWReader("10.10.10.121",9761);

#region 打开模块

Result result = jwReader.RFID\_Open();

if (result == Result.OK)

Console.WriteLine("Open Module Success");

else

{

Console.WriteLine("Open Module Failure");

goto Exit;

}

#endregion

#region 配置模块

RfidSetting rs = new RfidSetting();

rs.AntennaPort\_List = new List<AntennaPort>();

AntennaPort ap = new AntennaPort();

ap.AntennaIndex = 1;//天线1

ap.Power = 22;//功率设为27

rs.AntennaPort\_List.Add(ap);

rs.GPIO\_Trigger\_List = null;

rs.Inventory\_Time = 5;

rs.Region\_List = RegionList.CCC;

rs.RSSI\_Filter = new RSSIFilter();

rs.RSSI\_Filter.Enable = true;

rs.RSSI\_Filter.RSSIValue = (float)-70;

rs.Speed\_Mode = SpeedMode.SPEED\_FASTEST;

rs.Tag\_Group = new TagGroup();

rs.Tag\_Group.SessionTarget = SessionTarget.A;

rs.Tag\_Group.SearchMode = SearchMode.DUAL\_TARGET;

rs.Tag\_Group.Session = Session.S0;

result = jwReader.RFID\_Set\_Config(rs);

if (result == Result.OK)

Console.WriteLine("RFID Config Set Success");

else

{

Console.WriteLine("RFID Config Set Failure");

goto Exit;

}

#endregion

#region 读

Console.WriteLine("Start Read");

AccessParam accessParam = new AccessParam();

accessParam.Bank = MemoryBank.EPC;

accessParam.OffSet = 0;

accessParam.Count = 12;

for (int i = 1; i <= 10; i++)

{

string tagData = "";

result = jwReader.RFID\_Read(accessParam, out tagData);

if (result == Result.OK)

Console.WriteLine(string.Format("Count={0},EPC={1}",i,tagData));

else

Console.WriteLine("Read Failure");

}

Console.WriteLine("Stop Read");

#endregion

Exit:

#region 关闭模块

result = jwReader.RFID\_Close();

if (result == Result.OK)

Console.WriteLine("Close Module Success");

else

{

Console.WriteLine("Close Module Failure");

}

#endregion

Console.ReadLine();

}

}

}

## 写操作例程完整代码

using System;

using System.Collections.Generic;

using System.Text;

using JW.UHF;

namespace Write

{

class Program

{

static void Main(string[] args)

{

JWReader jwReader = new JWReader("COM3");

//JWReader jwReader = new JWReader("10.10.10.121",9761);

#region 打开模块

Result result = jwReader.RFID\_Open();

if (result == Result.OK)

Console.WriteLine("Open Module Success");

else

{

Console.WriteLine("Open Module Failure");

goto Exit;

}

#endregion

#region 配置模块

RfidSetting rs = new RfidSetting();

rs.AntennaPort\_List = new List<AntennaPort>();

AntennaPort ap = new AntennaPort();

ap.AntennaIndex = 1;//天线1

ap.Power = 10;//功率设为27

rs.AntennaPort\_List.Add(ap);

rs.GPIO\_Trigger\_List = null;

rs.Inventory\_Time = 5;

rs.Region\_List = RegionList.CCC;

rs.RSSI\_Filter = new RSSIFilter();

rs.RSSI\_Filter.Enable = true;

rs.RSSI\_Filter.RSSIValue = (float)-70;

rs.Speed\_Mode = SpeedMode.SPEED\_FASTEST;

rs.Tag\_Group = new TagGroup();

rs.Tag\_Group.SessionTarget = SessionTarget.A;

rs.Tag\_Group.SearchMode = SearchMode.DUAL\_TARGET;

rs.Tag\_Group.Session = Session.S0;

result = jwReader.RFID\_Set\_Config(rs);

if (result == Result.OK)

Console.WriteLine("RFID Config Set Success");

else

{

Console.WriteLine("RFID Config Set Failure");

goto Exit;

}

#endregion

#region 设置Criteria

RfidCriteria criteria = new RfidCriteria();

criteria.Bank = MemoryBank.EPC;

criteria.OffSet = 0;

criteria.Mask = Util.ToHexByte("3008");

criteria.Count = 2;

criteria.Match = true;

result = jwReader.RFID\_Set\_Criteria(criteria);

if (result == Result.OK)

Console.WriteLine("Set Criteria Success");

else

{

Console.WriteLine("Set Criteria Failure");

goto Exit;

}

#endregion

#region 写

Console.WriteLine("Start Write");

AccessParam accessParam = new AccessParam();

accessParam.Bank = MemoryBank.USER;

accessParam.OffSet = 0;

accessParam.Count = 2;

string writeData = "DCBA";

result = jwReader.RFID\_Write(accessParam, writeData);

if (result == Result.OK)

Console.WriteLine("Write Success");

else

Console.WriteLine("Write Failure");

#endregion

#region 读

Console.WriteLine("Start Read");

string readData = "";

result = jwReader.RFID\_Read(accessParam, out readData);

if (result == Result.OK)

Console.WriteLine(string.Format("Read Data User={0}",readData));

else

Console.WriteLine("Read Failure");

#endregion

#region 清除 Criteria

result = jwReader.RFID\_Clear\_Criteria();

if (result == Result.OK)

Console.WriteLine("Clear Criteria Success");

else

{

Console.WriteLine("Clear Criteria Failure");

goto Exit;

}

#endregion

Exit:

#region 关闭模块

result = jwReader.RFID\_Close();

if (result == Result.OK)

Console.WriteLine("Close Module Success");

else

{

Console.WriteLine("Close Module Failure");

}

#endregion

Console.ReadLine();

}

}

}